Die Beauftragte der Bundesregierung für Informationstechnik

XÖV

Freie Hansestadt Bremen

Handbuch zur Entwicklung XÖV-konformer IT-Standards

Version 1.1 vom 31. März 2012

Druckdatum 12. April 2012

Inhaltsverzeichnis

Zur Version 1.1 des Handbuches v

1 Einleitung 1

1.1 Standardisierung zwischen Freiheit und Bindung ... 2

1.2 XÖV-Konformität ... 3

1.3 Lebenszyklus eines XÖV-Standards ... 4

1.4 XÖV-Methoden und XÖV-Werkzeuge ... 6

1.5 Änderungsmanagement ... 7

1.6 Struktur des XÖV-Handbuches ... 7

1.7 Ansprechpartner ... 10

1.8 Mitwirkende ... 10

2 XÖV-Konformitätskriterien 13

2.1 Hinweis zur Prüfung der Konformität ... 13

2.2 Übersicht über die Kriterien ... 14

2.3 Detaillierte Beschreibung der Kriterien ... 15
Bereitstellungspflichten ... 15
Auskunftspflichten der Standardentwickler und -betreiber ... 17
Technische Kriterien ... 17

3 Produktion von XÖV-Standards 21

3.1 XÖV-Produktionszubehör der XÖV-Koordination ... 21

3.2 Produktionsumgebungen der XÖV-Vorhaben ... 23

3.3 XÖV-Produktionsprozess ... 24

4 XÖV-UML-Profil 25

4.1 Allgemeiner Aufbau eines UML-Profils ... 27

4.2 XÖV-Stereotypen ... 27
xsdAnyContents ... 28
xsdAttribute ... 29
xsdChoice ... 30
xsdCode ... 30
xsdCodeList ... 31
xsdCodeListEntry ... 32
xsdDeprecated ... 33
xsdDraft ... 34
xsdElement ... 34

xsdGlobalElement ... 36
xsdGlobalElementType ... 36
xsdGroup ... 37
xsdImport ... 37
xsdInclude ... 38
xsdKey ... 39
xsdKeyref ... 40
xsdLocalStructure ... 41
xsdMessage ... 42
xsdNamedType ... 43
xsdRestriction ... 44
xsdSchema ... 47
xsdTitled ... 49
xsdUnion ... 49
xsdUnionMember ... 50
xsdUnique ... 50
xsdWithImplementationHint ... 51
xsdXModel ... 51
xsdXModelImport ... 53

4.3 XÖV-Basisdatentypen ... 54
W3C-Datentypen ... 54
Datentyp Code ... 56
Datentyp String.Latin ... 59

5 XÖV-Namens- und Entwurfsregeln 61

5.1 Übersicht über die Regeln und Empfehlungen ... 61

5.2 Detaillierte Beschreibung der Regeln und Empfehlungen ... 63
Strukturen und Inhalte ... 63
Namen ... 68
Dokumentation ... 72
Wiederverwendung ... 73
Technik und Infrastruktur ... 76

6 Leitlinien zu Codelisten 79

6.1 Bereitstellung von Codelisten ... 79
Veröffentlichung und Dokumentation von Codelisten ... 80
Technisches Format für die Veröffentlichung von Codelisten ... 84
Rechtliche Bedingungen zur Veröffentlichung von Codelisten im XRepository ... 88

6.2 Anwendung von Codelisten in XÖV-Standards ... 89
Versionsrelevanz und Schemavalidierung ... 89
Darstellung der Einträge von Codelisten im XÖV-Standard ... 90
Einbindung von Codelisten in XÖV-Standards – Code-Datentypen ... 91
Spezielle Modellierungsmuster ... 98

7 Leitlinien zur Einbindung von XÖV-Kernkomponenten 101

8 Beispielhafte Umsetzung eines XÖV-Standards (XHamsterzucht) 103

8.1 Voraussetzungen ... 104

8.2 Definition der Anforderungen ... 104

8.3 Erstellung des XÖV-UML-Modells ... 105
Allgemeine Regeln ... 105
Aufbau der UML-Modell-Struktur ... 105
Abbildung der Anwendungsfälle ... 107
Abbildung von globalen Elementen ... 112
Abbildung der fachlichen Datentypen ... 114
Abbildung der Codelisten ... 118
Abbildung von Basisdatentypen ... 125
Darstellung von Wildcard-Elementen ... 126
Vereinfachte Darstellung von XML-Instanz-Inhalten ... 127
Ergänzungen zur Dokumentation ... 127
Übersicht der Pakete ... 128

9 XGenerator 131

9.1 Der XGenerator und sein Zubehör ... 132
XGenerator ... 132
XGenerator-Zubehör: Invarianten, Vorlagen und Operationen ... 133

9.2 Anwendung des XGenerators ... 136
Vorbereitung für die Anwendung des XGenerators ... 136
Erstellung eines Projekts im XGenerator ... 140
Automatisierte Erstellung von XML-Schema-Dateien und DocBook-Fragmenten ... 145

A Anhang zum Basisdatentyp String.Latin 155

B Überblick zu Velocity und OCL 157

B.1 Velocity ... 157

B.2 Object Constraint Language (OCL) ... 158

C XÖV-Glossar 159

D Versionshistorie 175

D.1 Release 1.1 ... 175

Zur Version 1.1 des Handbuches

Die erste Version des XÖV-Handbuches wurde im März 2010 herausgegeben. Der Kooperationsausschuss Bund-Länder-Kommunaler Bereich (KoopA ADV) hatte diese Version im Rahmen seiner letzen Sitzung verabschiedet und den Einsatz des Handbuches empfohlen. Seither ist es in vielen unterschiedlichen XÖV Vorhaben eingesetzt worden. Mehrere Standards der öffentlichen Verwaltung sind auf der Grundlage der in diesem Dokument genannten Kriterien bezüglich ihrer XÖV-Konformität zertifiziert worden. Eine Übersicht finden Sie auf der Webseite xoev.de.

Mit In-Kraft-Treten des Staatsvertrags zur Ausführung von Artikel 91c Grundgesetz (IT-Staatsvertrag) zum 1. April 2010 haben die darin vereinbarten Abstimmungsmechanismen die bisherigen Gremien abgelöst und sind in deren Rechtsnachfolge eingetreten. Seither ist der IT Planungsrat das zentrale Steuerungsgremium für die IT von Bund und Ländern. Im Rahmen dieser Veränderungen haben die E-Government-Staatssekretäre beschlossen, die Koordinierungsstelle für IT-Standards (KoSIT) in Bremen einzurichten. Die OSCI-Leitstelle, die bisher im Rahmen der XÖV-Koordination für das XÖV Handbuch zuständig war, ist zum 1. 4. 2011 in die KoSIT überführt worden. Die KoSIT gibt das XÖV Handbuch im Auftrag des IT-Planungsrat heraus.

Die hiermit vorgelegte Version 1.1 des Handbuches nimmt Änderungsanforderungen aus der Praxis der XÖV Vorhaben auf. Die Veränderungen gegenüber der vorherigen Version sind überschaubar und im Anhang Abschnitt D auf Seite 175 detailliert dargestellt.

1. EINLEITUNG

Die Umsetzung elektronisch unterstützter und medienbruchfreier Prozesse über alle Ebenen der öffentlichen Verwaltung hinweg ist abhängig von der Interoperabilität der eingesetzten informationstechnischen Systeme. Die Verwendung harmonisierter Nachrichten zur elektronischen Datenübertragung erhöht die Interoperabilität informationstechnischer Systeme. In Übereinstimmung mit dem European Interoperability Framework (EIF) definieren wir *Interoperabilität* wie folgt: *"die Fähigkeit verschiedener und unterschiedlicher Organisationen zur Interaktion zum beiderseitigen Nutzen und im Interesse gemeinsamer Ziele; dies schließt den Austausch von Informationen und Wissen zwischen den beteiligten Organisationen durch von ihnen unterstützte Geschäftsprozesse mittels Datenaustausch zwischen ihren jeweiligen IKT-Systemen ein."*

XÖV steht für XML *(Extensible Markup Language)* in der Öffentlichen Verwaltung und damit für Bestrebungen, die vorhandenen IT-Verfahren stärker als bisher zu vernetzen. Dies soll nicht zu Lasten der bestehenden Produktvielfalt gehen. Daher kommt der Standardisierung eine besondere Rolle bei der Verbesserung der Interoperabilität zu.

Ein *Standard* im Kontext der XÖV-Standardisierung ist eine Spezifkation von Nachrichten für den elektronischen Datenaustausch innerhalb und mit der öffentlichen Verwaltung. Ein *XÖV-Standard* ist ein Standard, dessen XÖV-Konformität die XÖV-Koordination bestätigt hat. XÖV-Standards sind offene und lizenzkostenfreie Standards, die allen Interessierten frei zugänglich zur Verfügung stehen. Die Nutzer von XÖV-Standards sind Entwickler oder Betreiber von informationstechnischen Systemen, die über Schnittstellen verfügen, die zum jeweiligen XÖV-Standard kompatibel sind.

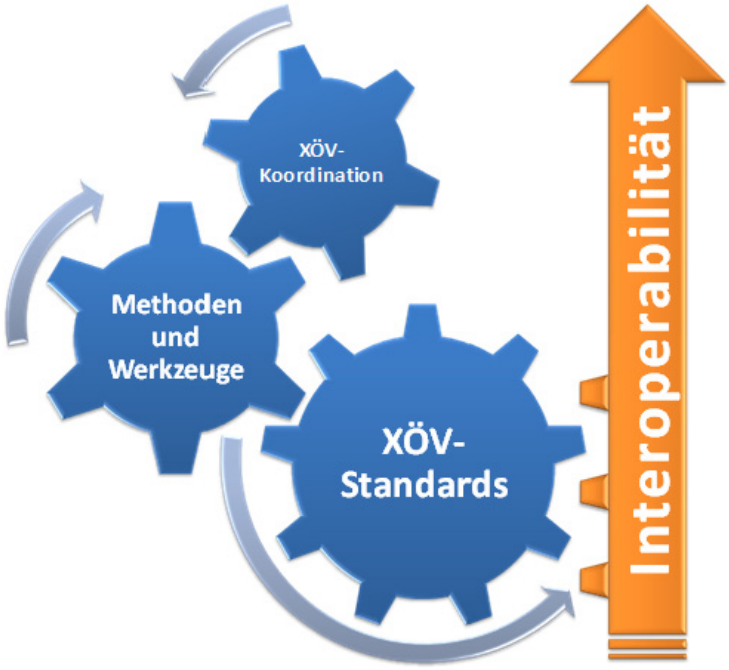
Mit der XÖV-Standardisierung sollen folgende Ziele erreicht werden:

- Erhöhung der Interoperabilität der informationstechnischen Systeme in der öffentlichen Verwaltung.
- Reduzierung der Risiken sowie der Koordinations- und Abstimmungskosten bei der Planung und Durchführung von IT-Projekten in der öffentlichen Verwaltung.
- Senkung von Transaktionskosten und Verbesserung der Qualität der übermittelten Daten bei elektronisch unterstützten Prozessen in der öffentlichen Verwaltung.

Die XÖV-Koordination dient der zentralen Harmonisierung und Unterstützung der XÖV-Vorhaben. Sie wird im Auftrag des IT-Planungsrat durch die Koordinierungsstelle für IT-Standards wahrgenommen. Durch die XÖV-Koordination wird sichergestellt, dass Erfahrungen und Erkenntnisse kontinuierlich in die Entwicklung zurückfließen und so allen anderen XÖV-Vorhaben zur Verfügung stehen. Dadurch wird die Wiederholung von Fehlern weitestgehend vermieden. Die Kosten für die Entwicklung neuer XÖV-Standards sinken durch die Reduktion von Koordinations-, Abstimmungs-, Fehler- und Fehlerfolgekosten sowie durch die Erhöhung der Anzahl der wiederverwendbaren Artefakte.

Das hiermit in der Version 1.1 vorgelegte XÖV-Handbuch umfasst alle für ein XÖV-Vorhaben relevanten Informationen. Es ist modular aufgebaut. Der wichtigste Aspekt des XÖV-Handbuches sind die Kriterien für die Prüfung der XÖV-Konformität eines Standards. Neben den XÖV-Konformitätskriterien stellt das XÖV-Handbuch zahlreiche Hinweise zum praktischen Umgang mit den technischen Aspekten der XÖV-Standardisierung bereit.

Bild 1-1 XÖV-Standardisierung und Interoperabilität



Das XÖV-Handbuch wendet sich an alle, die an der Entwicklung und Pflege eines XÖV-Standards mitwirken. Dazu gehören Mitarbeiter der öffentlichen Verwaltung, die mit der Planung, Konzeption, Implementierung und dem Betrieb von informationstechnischen Systemen für den elektronischen Datenaustausch innerhalb und mit der öffentlichen Verwaltung betraut sind. Insbesondere für die Verantwortlichen und Mitarbeiter von Projekten gibt das XÖV-Handbuch durch das beschriebene Regelwerk und die Hinweise zur technischen Realisierung eine zuverlässige Orientierung bei der Entwicklung von XÖV-Standards.

1.1 Standardisierung zwischen Freiheit und Bindung

Für unterschiedliche fachliche Domänen der öffentlichen Verwaltung gelten naturgemäß unterschiedliche fachliche Anforderungen an den Austausch von Daten. Zu den fachlichen Vorgaben können neben den Rechtsgrundlagen für die Datenübermittlung auch wirtschaftliche Rahmenbedingungen, vorhandene IT-Verfahren und deren Schnittstellen, Datenbestände in vorhandenen Registern, spezifische Vorgaben über die Registerführung, Anforderungen des Bestandsschutzes oder Ähnliches gehören.

Ein Standard muss der jeweiligen Fachlichkeit und weiteren Anforderungen seiner Domäne vollständig entsprechen, um den gewünschten Nutzen zu erbringen. Eventuelle Anpassungen der Arbeitsorganisation oder vorhandener Fachverfahren aufgrund des Standards dürfen die inhaltliche Aufgabenerfüllung nicht beeinträchtigen. Dies gilt auch für die Ausrichtung von XÖV-Standards an übergeordneten Vorgaben.

Gleichwohl sollen bei der Entwicklung fachlicher Standards für automatisierte Datenübermittlungen Optimierungspotenziale durch weitgehende Vereinheitlichung genutzt werden, wo immer die Aufgabenwahrnehmung es erlaubt. Es kommt dabei insbesondere darauf an, Unterschiede zwischen XÖV-Standards auf das durch fachliche Vorgaben bestimmte, notwendige Minimum zu reduzieren.

Zudem sollen die Standards so entwickelt und gepflegt werden, dass sie die von der Verwaltung bereitgestellten Infrastrukturkomponenten für die sichere und zuverlässige Datenübermittlung optimal nutzen. Entwicklungs-, Pflege- und Umsetzungsaufwände, die von der öffentlichen Verwaltung zu tragen sind, sollen dadurch nachhaltig reduziert werden.

Zur Erreichung der Ziele *Wirtschaftlichkeit, Risikominimierung, Qualitätsverbesserung* und *Interoperabilität* können XÖV-Vorhaben daher auf folgende Maßnahmen zurückgreifen:

- Nutzung der zentral zur Verfügung gestellten Methoden und Werkzeuge für die Entwicklung und die Pflege von XÖV-Standards,
- Wiederverwendung vorhandener fachlicher Lösungen, und
- Nutzung vorhandener Infrastrukturkomponenten der öffentlichen Verwaltung für die sichere und zuverlässige Datenübermittlung.

Aus unterschiedlichen fachlichen Vorgaben werden Unterschiede zwischen XÖV-Standards resultieren. Wir gehen davon aus, dass die Notwendigkeit des interdisziplinären Datenaustausches zwischen fachlichen Domänen der öffentlichen Verwaltung zunehmen wird. Daraus kann sich für Hersteller und Betreiber involvierter IT-Verfahren die Notwendigkeit ergeben, mehrere Schnittstellen für unterschiedliche XÖV-Standards parallel zu implementieren, zu betreiben und zu pflegen.

Unterschiede zwischen XÖV-Standards führen in diesen Fällen stets zu Aufwänden auf Seiten der Entwickler und Betreiber betroffener IT-Verfahren, sowie zu einem höheren Fehlerrisiko bei der ggf. erforderlichen Konvertierung der Daten in das für die jeweiligen Domänen vorherrschende Datenformat.

Solche Unterschiede und die damit verbundenen Aufwände und Risiken sind unvermeidlich, sofern ihnen fachliche Vorgaben zugrunde liegen. Sie sollen aber vermieden werden, sofern die Unterschiede zwischen XÖV-Standards *“nur”* aus der Unkenntnis bereits bestehender Lösungen, mangelnder Abstimmung, unterschiedlichen Entwicklungswerkzeugen oder individuellen Vorlieben der Entwickler herrühren.

1.2 XÖV-Konformität

Die Prüfung der XÖV-Konformität von Standards für den elektronischen Datenaustausch soll die Interoperabilität der XÖV-Standards untereinander verbessern. Durch die angestrebte XÖV-Konformität kann für XÖV-Vorhaben bei der Entwicklung eines Standards ein höherer Aufwand resultieren. Sobald jedoch unterschiedliche Fachgebiete miteinander kommunizieren, bietet die Verwendung fachunabhängiger (XÖV-Codelisten) und fachübergreifender Bausteine (XÖV-Kernkomponenten) erhebliche Vorteile bei der Entwicklung und der Pflege von fachspezifischen Standards für den elektronischen Datenaustausch.

Ein XÖV-Projekt, dessen Standard die XÖV-Konformitätskriterien erfüllt, kann zusätzlich von folgenden Nebeneffekten profitieren: gesteigerte Wirtschaftlichkeit und reduzierte Risiken bei der Entwicklung sowie eine hohe Qualität des entwickelten Standards.

Die XÖV-Konformitätskriterien wurden nach folgenden Grundsätzen gestaltet:

- Sie verbessern die Interoperabilität aller XÖV-Standards.
- Sie sind adressatengerecht dargestellt, transparent und handhabbar.
- Sie behindern nicht die Umsetzung von fachspezifischer Anforderungen in einen XÖV-Standard.
- Sie lassen weit gehende Freiheiten bezüglich des Vorgehens und der Organisation eines XÖV-Projektes.
- Sie treffen Aussagen über die formale Beschaffenheit eines *Standards* für den elektronischen Datenaustausch und seine Eignung für technische Interoperabilität. Sie treffen *keine* Aussagen über die fachliche Qualität sowie den Nutzen des *XÖV-Vorhabens*, in dessen Rahmen der Standard entwickelt wird.

Die Grundlage für die Prüfung der XÖV-Konformität eines Standards sind die XÖV-Konformitätskriterien. Sie sind als Muss-, Soll- und Kann-Regeln definiert und in drei Gruppen unterteilt: Informationspflichten (XÖV-Steckbrief) und Bereitstellungspflichten (XÖV-UML-Modell, XML Schema, Dokumentation) sowie technische Anforderungen.

Der Schwerpunkt der Prüfung von Standards auf XÖV-Konformität liegt bei der Prüfung der Erfüllung der technischen Anforderungen. Die Muss- und Soll-Regeln der technischen Anforderungen sind in das *XÖV-Produktionszubehör* integriert. Daher kann die Prüfung der Erfüllung der technischen Anforderungen weitgehend automatisiert erfolgen. Beachten Sie bitte die Hinweise zur Prüfung der XÖV-Konformität in Abschnitt 2.1 auf Seite 13.

Die XÖV-Konformitätskriterien regeln auch, in welcher Form ein Standard erstellt und vorgelegt werden muss, damit dieser auf die Erfüllung der technischen Anforderungen überprüft werden kann.

Darüber hinaus legen die XÖV-Konformitätskriterien die Nutzung von bereits vorhandenen fachübergreifenden und fachunabhängigen Bausteinen fest.

1.3 Lebenszyklus eines XÖV-Standards

Im Kontext der XÖV-Standardisierung spielt die Definition eines Lebenszyklus eine zentrale Rolle, da aus dem Zustand des Standards unterschiedliche Aufgaben resultieren. Insbesondere die Erfüllung der Informationspflichten im Sinne des XÖV-Handbuchs unterstützt durch den XÖV-Steckbrief ist davon abhängig, in welcher Phase des Lebenszykluses sich ein Standard befindet.

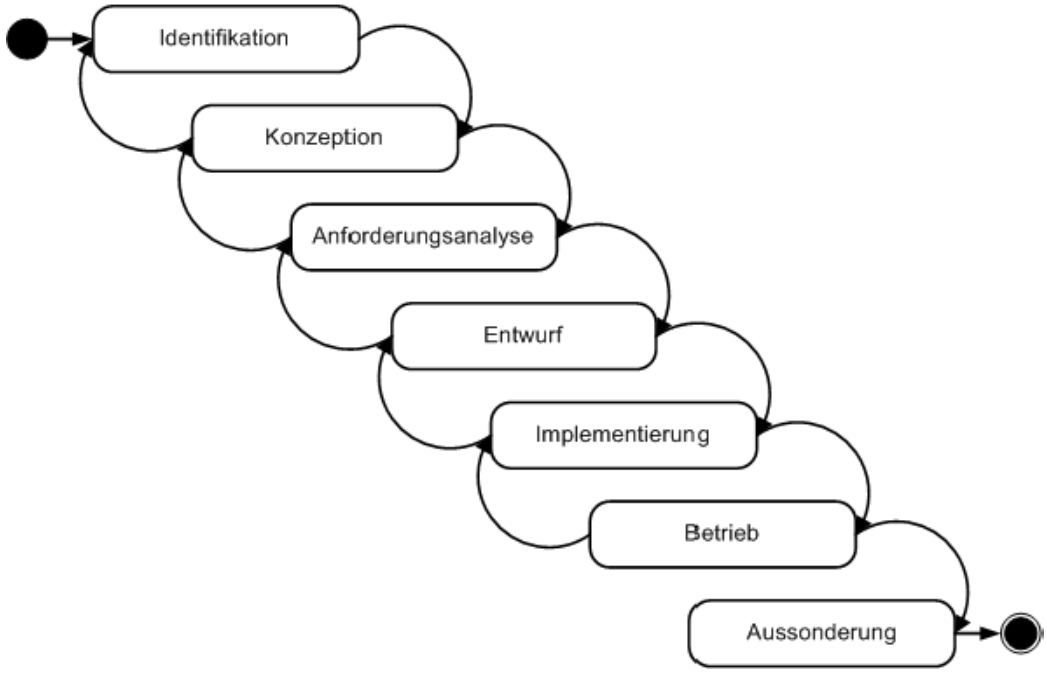
Nachfolgend wird das Modell eines Lebenszyklus basierend auf *Generalised Enterprise Reference Architecture and Methodology* (GERAM) [1] anhand von Phasen beschrieben.

Der Lebenszyklus eines Standards gliedert sich in die sieben Phasen *Identifikation, Konzeption, Anforderungsanalyse, Entwurf, Implementierung, Betrieb* und *Aussonderung*.

Ein *XÖV-Vorhaben* umfasst den gesamten Lebenszyklus eines Standards. Ein *XÖV-Projekt* umfasst die Phasen Anforderungsanalyse, Entwurf und Implementierung eines Standards. Der beschriebene Lebenszyklus legt kein Vorgehensmodell für XÖV-Vorhaben fest.

1.ISO 15704:2000 Industrial automation systems – Requirements for enterprise-reference architectures and methodologies

Bild 1-2 Lebenslauf von XÖV-Standards



- Innerhalb der Identifikationsphase wird der Bedarf festgestellt und eine erste Abgrenzung auch zu anderen, bereits existierenden XÖV-Vorhaben durchgeführt. Sie schließt in der Regel mit einer Definition des Gegenstandes des XÖV-Vorhabens ab.
- Ihr folgt die Konzeptionsphase, in der der Standard durch seine Ziele, Visionen und Umsetzungskonzepte beschrieben wird. In dieser Phase werden auch Entscheidungen über die Wiederverwendung von fachübergreifenden und fachunabhängigen Bausteinen getroffen.
- Die Festlegung der funktionalen und nicht funktionalen Anforderungen an den Standard erfolgt in der Phase Anforderungsanalyse.
- In der Entwurfsphase werden die Anforderungen an den Standard in ein UML-Modell überführt.
- Die Umsetzung des UML-Modells des Standards in XML Schema Dateien und die Fertigstellung der Dokumentation erfolgt in der Phase Implementierung.
- In der Betriebsphase wird der Standard in informationstechnischen Systemen verwendet. Weiterhin wird in dieser Phase die dauerhafte Verwendbarkeit des Standards durch Prozesse zum Problem-, Änderungs- und Konfigurationsmanagement sichergestellt.
- Maßnahmen die dazu dienen den Standard aus dem Betrieb zu lösen, werden in der Aussonderungsphase durchgeführt.

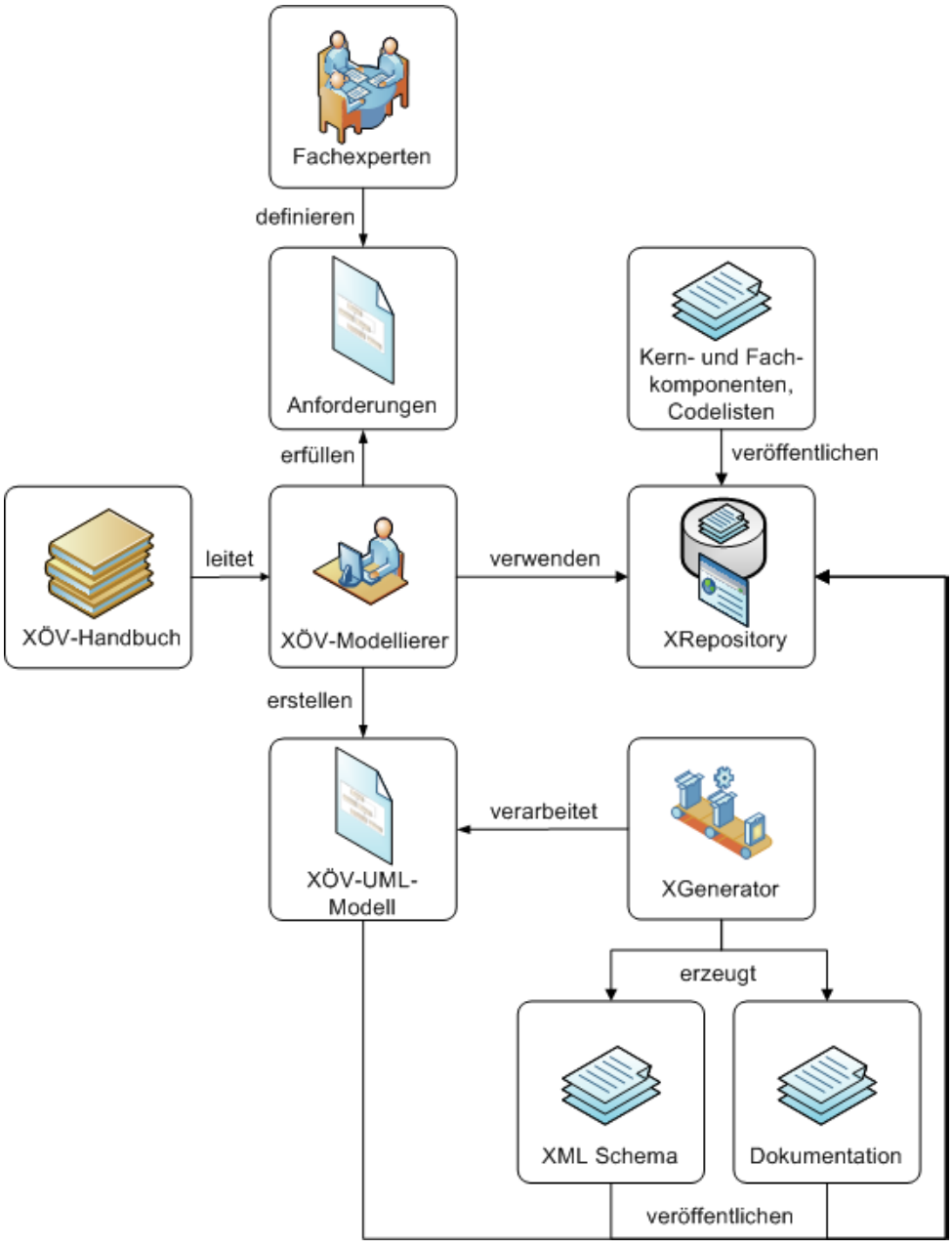
Bei der Durchführung eines XÖV-Vorhabens können mehrere Lebenszyklusphasen zusammenfallen.

In der Betriebsphase kann es zu Situationen kommen, in denen sich die Anforderungen an den Standard ändern und ein XÖV-Projekt zur Weiterentwicklung des Standards notwendig wird. Der Standard durchläuft dann die vorhergehenden Lebenszyklusphasen erneut.

1.4 XÖV-Methoden und XÖV-Werkzeuge

Die XÖV-Methoden und XÖV-Werkzeuge für die Entwicklung und Bereitstellung von XÖV-Standards basieren auf internationalen Standards [1]. Sie wurden im Rahmen des Deutschland-Online Vorhaben *Standardisierung* unter Mitwirkung von Vertretern der XÖV-Vorhaben erstellt und dienen als Leitfaden für die Durchführung von XÖV-Vorhaben sowie der Vereinfachung der Wiederverwendung von fachunabhängigen und fachübergreifenden Bausteinen.

Bild 1-3 Infrastruktur der XÖV-Standardisierung



1.ISO/IEC Open-EDI Reference Model, OMG Unified Modeling Language (UML), UN/CEFACT Core Components Technical Specification (CCTS), W3C Extensible Markup Language (XML), W3C XML Schema Definition Language (XSD)

Der XGenerator ist eine frei verfügbare Software, mit der automatisiert XÖV-UML-Modelle überprüft und in XML Schema Dateien transformiert werden können. Weiterhin kann der XGenerator aus XÖV-UML-Modellen Dokumentation im Format DocBook XML[1] generieren. DocBook XML Dateien können automatisiert in verschiedene andere Dokumentenformate transformiert werden. Der XGenerator stellt die einheitliche Erzeugung von XÖV-Standards, ihrer Dokumentation sowie die Einhaltung der technischen XÖV-Konformitätskriterien sicher. Die Kombination des XGenerator, des XÖV-Produktionszubehörs und weiterer Werkzeuge zur Produktion von XÖV-Standards wird als Produktionsumgebung bezeichnet.

Das XRepository ist eine öffentliche, über das Internet zugängliche und zentral verwaltete Bibliothek für XÖV-Standards, XÖV-Kernkomponenten, XÖV-Fachkomponenten sowie XÖV-Codelisten. Das XRepository vereinfacht die Publikation und Recherche sowie die Wiederverwendung vorhandener Ergebnisse der XÖV-Standardisierung.

1.5 Änderungsmanagement

Änderungsanträge sind ein zentrales Instrument zur Entwicklung der XÖV-Methodik und den damit einhergehenden Werkzeugen. Sie helfen bei der kontinuierlichen Verbesserung der Qualität existierender Lösungen und bieten den erforderlichen Rahmen zur Weiterentwicklung konzeptioneller Ansätze. Treibende Kraft ist in beiden Fällen die wertvolle Erfahrung, die mit den XÖV-Methoden und XÖV-Werkzeugen im praktischen Einsatz gemacht werden.

Änderungsanträge können prinzipiell durch alle Personen gestellt werden, die direkt oder indirekt an der Entwicklung oder Pflege einer XÖV-Spezifikation beteiligt sind. Neben einem beschreibenden Text sollte ein Änderungsantrag Informationen zum Antragsteller (Name, Organisation und Email-Adresse) und dem Kontext des Antrags enthalten.

Änderungsantrage können an die KoSIT (kosit@finanzen.bremen.de) gerichtet werden. Mit der Erfassung des Antrags übernimmt die KoSIT die Koordination der weiteren Bearbeitung. Das Ergebnis der Bearbeitung wird dem Antragsteller durch die KoSIT mitgeteilt.

Leider lassen die derzeitigen Strukturen zur Bearbeitung von Änderungsantragen weder die direkte Einbindung des Antragstellers in den Prozess der Bearbeitung noch die Veröffentlichung bestehender Änderungsanträge zu. Es ist geplant, diese Situation in Zukunft zu verbessern.

1.6 Struktur des XÖV-Handbuches

Durch die XÖV-Koordination werden an jedes XÖV-Vorhaben, welches einen XÖV-konformen Standard entwickeln oder betreiben möchte, Anforderungen gestellt deren Erfüllung überprüft werden kann. Gleichzeitig stellt die XÖV-Koordination Hilfsmittel und Werkzeuge zur Verfügung, die den XÖV-Vorhaben helfen können, die Anforderungen zu erfüllen. In diesem Sinne des *Fordern und Fördern* ist auch das vorliegenden Handbuch aufgebaut. Es nennt als Forderung die Kriterien, die zu erfüllen sind, und es fördert durch Bereitstellung der Methoden und Werkzeuge für eine effiziente Entwicklung XÖV-konformer Standards.

1.In der Version 4.1.2 (DTD)

In dem Kapitel “XÖV-Konformitätskriterien” auf Seite 13 sind die Kriterien genannt, die bei der Prüfung auf XÖV-Konformität zu Grunde gelegt werden. Einige der technischen Kriterien nehmen Bezug auf Detailregelungen, die in späteren Abschnitten genauer ausgeführt werden. So lautet eines der Konformitätskriterien, dass bestimmte *“Namens- und Entwurfsregeln”* beachtet werden müssen. Diese zum Teil sehr technischen Regelungen sind im Abschnitt “XÖV-Namens- und Entwurfsregeln” auf Seite 61 im Detail ausgeführt. Sie sind bereits in der Modellierungsphase zu beachten. Ohne automatisierte Unterstützung ist dies aufwändig. Daher wurden die Regeln so formuliert, dass sie in technischen Verfahen umgesetzt und automatisiert überprüft werden können. In dem Abschnitt “Produktion von XÖV-Standards” auf Seite 21 wird das Produktionszubehör vorgestellt, das im Rahmen der Entwicklung und des Betriebes XÖV-konformer Standards genutzt werden kann und soll. Sie stehen insbesondere den XÖV-Vorhaben zur Verfügung und sollen herangezogen werden, um von Beginn an die Regularien für XÖV-Konformität zu berücksichtigen. Eine besondere Rolle nimmt dabei die Software *XGenerator* ein. Sie basiert auf der Entwicklungsumgebung Eclipse und dem OCL-Werkzeug USE[1]. Die Software *XGenerator* soll sowohl von den XÖV-Vorhaben im Rahmen der Entwicklung und Pflege der XÖV-Standards, als auch von den Organisationseinheiten, welche die XÖV-Konformität prüfen, herangezogen werden. Da diese Software eine so zentrale Rolle spielt, ist ihr der eigene Abschnitt “XGenerator” auf Seite 131 gewidmet

Der gesamte Entwicklungsprozess eines XÖV-Standards basiert auf dem Grundgedanken, dass die Anwendungsfälle, die Prozesse und die Datenstrukturen eines Datenaustausches zunächst in verschiedenen Abstraktionsebenen formalisiert beschrieben werden. Diese Beschreibung erfolgt in dem international üblichen Format UML *(Unified Modelling Language)*. Anschließend erfolgt eine *“Übersetzung”* in die maschinell interpretierbare Beschreibung der Nachrichten- und Datenstrukturen in dem Format *XML*. Damit diese *“Übersetzung”* funktioniert, muss UML auf eine bestimmt Art und Weise genutzt werden, nämlich zielgerichtet für den späteren Übersetzungsprozess. Zudem erlaubt UML im Allgemeinen die Beschreibung so ziemlich jedes Sachverhaltes, der sich irgendwie mit den Methoden der Informatik fassen lässt. Im Kontext der XÖV-Koordination geht es aber um den spezifischen Teilbereich des elektronischen Datenaustausches zwischen IT-Systemen unter Beachtung der Anforderungen der öffentlichen Verwaltung an Interoperabilität. Die Art und Weise, wie UML im Kontext der XÖV-Koordination genutzt werden soll, ist selbst wiederum in einem fachlichen Modell beschrieben. Dieses *Metamodell* beschreibt generisch die zu regelnden Aspekte von Datenaustauschszenarien, wenn man sie denn XÖV-konform umsetzen will. Es ist in dem “XÖV-UML-Profil” auf Seite 25 beschrieben. Es muss in UML-Modellierungswerkzeugen technisch implementiert werden, damit es dann anschließend bei der Entwicklung von Fachmodellen für den XÖV-konformen Datenaustausch zu Grunde gelegt werden kann.

Mit den Konformitätskriterien, dem Metamodell (repräsentiert durch das XÖV-Profil für UML) und den Namens- und Entwurfsregeln ist der formale Rahmen aufgestellt, innerhalb dessen die Entwicklung erfolgen muss. Mit den Werkzeugen und dem Zubehör wird der Entwicklungsprozess innerhalb dieses Rahmens unterstützt. Was im Sinne der Interoperabilität noch fehlt, sind *gemeinsame Datenstrukturen,* auf die sich die Kommunikationspartner beziehen können und sollen. Das ist der Aspekt der Wiederverwendung vorhandener Lösungen. Solche zentral bereitgestellten, an verschiedenen Stellen genutzten Komponenten kann es (im Kontext der XÖV-Koordination) in zwei Ausprägungen geben:

1.Sowohl Eclipse als auch USE sind unentgeltlich zu beziehen, ihr Quelltext liegt offen, und ihre Lizenzbestimmungen fördern die Weiterentwicklung.

Fachunabhängig Manche Komponenten sind so grundlegender und generischer Natur, dass sie grundsätzlich in allen Datenübermittlungsszenarien eingesetzt werden können und sollen. Ihr Einsatz ist unabhängig vom fachlichen Zusammenhang.

Fachunabhängige Basis-Datentypen werden in dem Abschnitt “XÖV-Basisdatentypen” auf Seite 54 beschrieben. Grundsätzlich werden im XÖV-Kontext die Datentypen von XML Schema Version 1.0 des W3C genutzt. Darüber hinaus kann es Basisdatentypen geben, die speziell für den XÖV-Kontext entwickelt werden. In der vorliegenden Fassung dieses Handbuches sind das zwei Datentypen. Zum einen der Datentyp *Code*. der genutzt werden soll um Werte aus vorher vereinbarten Codelisten zu übermitteln[1]. Codes und Codelisten spielen für die Interoperabilität im elektronischen Datenaustausch eine herausragende Rolle, und dem Umgang mit Codelisten ist deshalb der eigene Abschnitt “Leitlinien zu Codelisten” auf Seite 79 gewidmet.

Der zweite Basisdatentyp, der zur fachübergreifenden Interoperabilität beitragen wird, und der speziell für den XÖV-Kontext entwickelt worden ist, ist der Datentyp *String.Latin*, der mittlerweile als Bestandteil des eigenständigen Standards *“Lateinische Zeichen in Unicode”*[2] ausgeliefert wird. Interoperabilität zwischen IT-Systemen beginnt bereits bei dem verwendeten Zeichensatz, und bisher ist die Situation in Deutschland diesbezüglich alles andere als zufriedenstellend. Für die Datenübermittlung und — weitaus problematischer — auch für die Registerführung werden die unterschiedlichsten Zeichensätze genutzt. Um diese Situation zu verbessern wurde der genannte Standard im Kontext von XÖV eingeführt. Der Datentyp *String.Latin* schränkt die Menge der weltweit in Gebrauch befindlichen Zeichen auf die lateinischen Zeichen ein. Der Zeichensatz orientiert sich an rechtlichen Verpflichtungen ebenso wie an internationalen technologischen Entwicklungen sowie an der Verwaltungspraxis in Deutschland[3]. Der Datentyp ist technisch als Einschränkung des W3C-Datentypen für Zeichenketten realisiert.

Fachübergreifend Wiederverwendbare Komponenten, die in mehreren fachlichen Zusammenhängen auftreten, beispielsweise bei Datenübermittlungen in verschiedenen Fachressorts, sollen als *fachübergreifende Komponenten* modelliert und bereitgestellt werden. Sie erhalten einen zusätzlichen Nutzen im Rahmen der Datenübermittlung *zwischen Fachressorts*. Typische Kandidaten für solche komlexen Komponenten, die in unterschiedlichen Kontexten genutzt werden können, wären Datenstrukturen wie der *Name einer Person*, die *Anschrift* und andere.

Die Entwicklung und der Gebrauch solcher wiederverwendbarer Komponenten ist komplex, da die zentral bereitgestellten Lösungen fast immer an das konkrete fachliche Szenario angepasst (also: modifiziert)[4] werden müssen. Bei der Entwicklung wiederverwendbarer Komponenten im XÖV-Kontext hat man sich an dem Konzept für Kernkomponenten *(Core Components)* orientiert, wie es von der UN/CEFACT im Rahmen der ebXML Initiative entwickelt worden ist. Im Rahmen der XÖV Koordination wurde die *XÖV-Kernkomponenten-Bibliothek* entwickelt und im XRepository bereit gestellt. Es handelt sich um eine Sammlung von wiederverwendbaren Objekten für den elektronischen Datenaustausch innerhalb der öffentlichen Verwaltung in Deutschland sowie zwi-

1.Zum Beispiel: Übermittlung des Codes `w` für den Wert `weiblich` in der vorher abgestimmten Codeliste der Geschlechter von Personen.

2.verfügbar unter http://xoev.de/latinchars/

3.So erfolgte u. B. eine Abstimmung mit der Bundesdruckerei bezüglich des bei der Ausstellung von hoheitlichen Dokumenten genutzten Zeichensatzes.

4.So benötigt man zum Beispiel im Meldewesen bestimmte Komponenten einer postalischen Anschrift nicht, die in anderen Szenarien unverzichtbar sind.

schen der öffentlichen Verwaltung und ihren Kunden. Die XÖV-Kernkomponenten (Version 1.0) wurden durch den KoopA ADV abgenommen und herausgegeben. Sie wird derzeit in den XÖV-Vorhaben erprobt.

Die bisher vorliegenden Ergebnisse haben gezeigt, dass an bestimmten Stellen Änderungsbedarf bezüglich des methodischen Umgangs mit den Kernkomponenten erforderlich sind. Daher werden Leitlinien für die Arbeit mit Kern- und Fachkomponenten zu entwickeln sein. Das Thema konnte noch nicht abschließend bearbeitet werden. Der Abschnitt “Leitlinien zur Einbindung von XÖV-Kernkomponenten” auf Seite 101 ist in der vorliegenden Fassung des XÖV-Handbuches deshalb nur ein *“Platzhalter”*.

Die besten Konzepte nutzen nichts, wenn sie nicht richtig angewandt werden. Die Konzepte der XÖV-konformen Entwicklung von Standards müssen an deren Nutzer und Anwender vermittelt werden, also die *Leiter von XÖV-Vorhaben* und die *Entwickler von XÖV Standards*. Als Beginn der erforderlichen Vermittlungs- und Schulungsmaßnahmen kann ein Beispielprojekt dienen, wie es in dem Abschnitt “Beispielhafte Umsetzung eines XÖV-Standards (XHamsterzucht)” auf Seite 103 vorgestellt wird. Anhand eines fiktiven Beispiels[1] wird ein XÖV-Vorhaben möglichst vollständig beschrieben. Diese Beispielprojekt wird auch im XRepository bereitgestellt.

1.7 Ansprechpartner

Das XÖV-Handbuch wird von der bei der Freien Hansestadt Bremen angesiedelten *Koordinierungsstelle für IT-Standards* (KoSIT) herausgegeben. Informationen rund um XÖV erhalten Sie auf unserer Webseite www.xoev.de. Bei Fragen und Rückmeldungen wenden Sie sich bitte an:

- Die Koordinierungsstelle für IT-Standards, EMail `Kosit@finanzen.bremen.de`

1.8 Mitwirkende

An der Version 1.0 des Handbuches haben folgende Personen mitgewirkt:

Name	Institution	eMail
Büttner, Dr. Fabian	TZI, MSI Unternehmensberatung	fabian.buettner@gmx.org
Cordes, Nils	BMI	nils.cordes@bmi.bund.de
Crome, Cornelia	BIT	cornelia.crome@bva.bund.de
Drees, Simon	KoSIT	simon.drees@finanzen.bremen.de
Franke, Antje	]init[AG	antje.franke@init.de
Hamann, Lars	TZI	lhamann@tzi.de
Heins, Jessica	KoSIT	jessica.heins@finanzen.bremen.de
Karich, Christoph	CSC	ckarich@csc.com
Krolczyk, Adrian	TU Berlin	adrian.krolczyk@sysedv.tu-berlin.de

1.Uns ist sehr wohl bewusst, dass die Komplexität der Deutschen Vereinsorganisation im Allgemeinen und der Aufzucht und Pflege mäuseartiger Wühler im Speziellen unzulässig reduziert worden ist. Es handelt sich um ein für didaktische Zwecke erstelltes Beispiel und sollte entsprechend milde hinsichtlich fachlicher Aussagen beurteilt werden.

Name	Institution	eMail
Kuhlmann, Mirco	TZI	mk@tzi.de
Lahmann, Karen	MSI Unternehmensberatung	k.lahmann@msi-beratung.de
Lange, Dr. Christian	BIT	christian.lange@bva.bund.de
Lopes, Dominique	]init[AG	dominique.lopes@init.de
Rabenstein, Yorck	]init[AG	yorck.rabenstein@init.de
Senf, Christian	TU Berlin	christian.senf@sysedv.tu-berlin.de
Steimke, Frank	KoSIT	frank.steimke@finanzen.bremen.de
Stosiek, Alina	]init[AG	alina.stosiek@init.de
Weber, Hannes	KoSIT	hannes.weber@finanzen.bremen.de
Zimmer, Dr. Wolf	CSC	wzimmer2@csc.com

Die Änderungen der Version 1.1 wurden durch die KoSIT auf Basis der Rückmeldungen und Änderungswünsche von Anwendern umgesetzt.

2. XÖV-KONFORMITÄTSKRITERIEN

Dieses Kapitel beschreibt die konkreten Prüfkriterien, die ein Standard erfüllen muss, um die XÖV-Konformität zu erlangen. Zu jedem Kriterium ist angegeben, welche Prüfgrundlage vom XÖV-Vorhaben zu liefern ist und welcher Prüfinhalt von der XÖV-Koordination bewertet wird.

Die nachfolgenden Kriterien sind den drei Bereichen

- Bereitstellungspflichten,
- Auskunftspflichten der Standardentwickler und -betreiber sowie
- Technische Kriterien

zugeordnet und werden in die drei Verbindlichkeitsstufen **Muss, Soll** und **Empfehlung** unterschieden.

- Die Verbindlichkeit **Muss** beschreibt die zwingend von einem Standard zu erfüllenden Kriterien für die Einstufung "XÖV-konform", um die Ziele der XÖV-Standardisierung erreichen zu können.
- Die Verbindlichkeit **Soll** ermöglicht Abweichungen von den Kriterien; jedoch sind diese der XÖV-Koordination zu begründen. Zu Abweichungen kann es z. B. dann kommen, wenn die im Abschnitt 1.1 auf Seite 2) genannten Rahmenbedingungen die vollständige Einhaltung der Kriterien (noch) nicht erlauben.
- Die **Empfehlungen** dienen der Nutzung einheitlicher und in der Praxis bewährter Muster und Praktiken, nach denen sich XÖV-Vorhaben richten können. Die Umsetzung der Empfehlungen kann den Aufwand bei der Entwicklung eigener Lösungen reduzieren und verbessert gleichzeitig die Interoperabilität zwischen den XÖV-Standards. Empfehlungen haben keine Relevanz für XÖV-Konformität.

2.1 Hinweis zur Prüfung der Konformität

Die Konformität eines Standard zu den Konformitätskriterien wird von der Bundesstelle für Informationstechnik (BIT) anhand der nachfolgend genannten Kriterien überprüft. Auf Basis eines Prüfberichtes und einer Empfehlung der BIT entscheidet die KoSIT über die XÖV-Zertifizierung. Die BIT ist damit die *“XÖV Zertifizierungsstelle”*.

Die XÖV-Zertifizierungsstelle bietet allen XÖV-Vorhaben die Möglichkeit, den entwickelten Standard durch Vergabe eines XÖV-Zertifikats zu einem XÖV-Standard zu zertifizieren. Für eine erfolgreiche XÖV-Zertifizierung werden verschiedene Projektergebnisse benötigt, die der XÖV-Zertifizierungsstelle vorzulegen und durch diese zu prüfen sind. Bei der BIT ist eine detaillierte *“Anleitung zur Zertifizierung eines Standard gemäß der XÖV Vorgaben”* erhältlich.

Für die Zertifizierung müssen verschiedene Artefakte eingereicht werden. Der wichtigste Bestandteil der Zertifizierung ist dabei das Fachmodell des Datenaustauschs, welches den zu prüfenden Standard umfassend definiert und den fachlichen Kontext der Datenübermittlung beschreibt. Das Fachmodell

muss in UML Version 2.x notiert sein (siehe K-9 auf Seite 18). Die inhaltliche und formale Prüfunge der XÖV-Zertifizierungsstelle basieren zu einem erheblichen Teil auf diesem Fachmodell. Viele formale Eigenschaften des Standard, insbesondere die Einhaltung der für XÖV-Standards einschlägigen Namens- und Entwurfsregeln (siehe Abschnitt 5 auf Seite 61), können automatisiert überprüft werden. Die XÖV-Zertifizierungsstelle nutzt dafür das XÖV-Produktionszubehör (siehe Abschnitt 3.1 auf Seite 21).

Das Fachmodell wird der XÖV-Zertifizierungsstelle in dem von der Object Management Group (OMG) definierten Format **XML Metadata Interchange (XMI)** übergeben. Das Format ist offen und anbieterneutral und gestattet den Datenaustausch von Objekten auf Basis von Meta-Metamodellen nach der Meta-Object Facility (MOF). Durch Nutzung dieses standardisierten Austauschformats für die in UML notierten Fachmodelle *soll* erreicht werden, dass beliebige UML-Werkzeuge genutzt werden können um das Fachmodell des Datenaustausches zu erstellen. Leider ist die Erfahrung der KoSIT, dass dieses Ideal noch nicht erreicht ist. XMI bietet derzeit noch zu viele Freiheitsgrade, so dass nicht immer davon ausgegangen werden kann, dass ein in UML notiertes und im Format XMI zur Zertifizierung eingereichtes Fachmodell eines designierten XÖV-Standard der automatisierten Prüfung durch die XÖV-Zertifizierungsstelle zugänglich ist. *Faktisch* besteht derzeit noch eine Abhängigkeit von den Produkten für die UML-Modellierung. Die Konformitätsregel K-10 sieht aus diesem Grunde ausdrücklich vor, dass das XÖV-UML-Modell als XMI-Repräsentation *in einer für den XGenerator lesbaren Form* eingereicht werden muss.

Ob die XMI-Repräsentation des Fachmodells für den XGenerator lesbar ist oder nicht, hängt nach dem derzeitigen Stand der Technik von dem UML-Werkzeug ab. Die OMG arbeitet unter dem Titel Canonical XMI an einer Verbesserung der Situation. Die KoSIT wird Ergebnisse in das XÖV Produktionszubehör übernehmen, sobald Produktionsreife erreicht worden ist. Bis dahin empfehlen wir XÖV Vorhaben, sich frühzeitig mit der XÖV-Zertifizierungsstelle bzw. der KoSIT in Verbindung zu setzen um Fragen der Konformität zu klären.

2.2 Übersicht über die Kriterien

Nr.	Verbindlichkeit	Kurzbeschreibung	Seite
Bereitstellungspflichten			
K-1	Muss	Ein Standard der öffentlichen Verwaltung	15
K-2	Muss	Freie Verwendung	15
K-3	Muss	Dokumentation	16
K-4	Muss	Veröffentlichung	16
K-5	Muss	Nachhaltigkeit des Standards	16
Auskunftspflichten der Standardentwickler und -betreiber			
K-6	Muss	Anzeige der Entwicklungsabsicht	17
K-7	Muss	Informationen zum Status quo des Standards	17
Technische Kriterien			
K-8	Soll	Modellierung der Prozesse in UML	17
K-9	Muss	Modellierung der Datenstrukturen in UML	18

Nr.	Verbindlichkeit	Kurzbeschreibung	Seite
K-10	Muss	Einhaltung der XÖV-Namens- und Entwurfsregeln	18
K-11	Soll	Nutzung von XÖV-Kern- und Fachkomponenten	19
K-12	Soll	Nutzung der XÖV-Basisdatentypen	19
K-13	Soll	Nutzung von Codelisten	19
K-14	Muss	Erfolgreiche Verarbeitung des XÖV-UML-Modells durch das XÖV-Produktionszubehör	19
K-15	Soll	Nutzung einer sicheren Infrastruktur für den elektronischen Datenaustausch	20

2.3 Detaillierte Beschreibung der Kriterien

2.3.1 Bereitstellungspflichten

Diese Kriterien klären, von wem und wie ein XÖV-Standard für die öffentliche Verwaltung bereitzustellen ist. Insbesondere werden die Mindestanforderungen an die Offenheit eines XÖV-Standards festgelegt.

K-1 (MUSS): Ein Standard der öffentlichen Verwaltung

"Eigentümerin" des XÖV-Standards muss die öffentliche Verwaltung sein, d.h. sie bestimmt seine Inhalte und hat alle Rechte am Standard inne. Weiter entscheidet sie über Entwicklung und Pflege sowie über die Verwendung des Standards.

Begründung:

Die Entscheidung der öffentlichen Verwaltung über ihre Prozesse soll nicht durch kommerzielle Abhängigkeiten geprägt werden.

Prüfgrundlage:

Selbstauskunft des Vorhabens im Steckbrief

Prüfinhalt:

Entscheidergremien sind von der öffentlichen Verwaltung besetzt

K-2 (MUSS): Freie Verwendung

Der XÖV-Standard muss frei von Rechten Dritter sein. Er muss innerhalb der öffentlichen Verwaltung der Bundesrepublik Deutschland und für die Nutzer ihrer Dienstleistungen uneingeschränkt und unentgeltlich verwendbar sein und bleiben.

Begründung:

Mit der freien Verfügbarkeit möchte man die Herstellerunabhängigkeit von Schnittstellen und deren Wiederverwendbarkeit fördern.

Prüfgrundlage:

Einstellung des Standards in das XRepository

Prüfinhalt:

alle Teile des Standards sind frei von Rechten Dritter

K-3 (MUSS): Dokumentation

Ein XÖV-Standard muss alle Informationen bereitstellen, die erforderlich sind, um eine standard-konforme Schnittstelle für IT-Verfahren zu entwickeln. Er ist in Form von XML-Schema-Dateien und deren konsistenter Dokumentation an seine Nutzer auszuliefern.

Begründung:

Eine standard-konforme Schnittstelle für IT-Verfahren kann nur (weiter-)entwickelt werden, wenn alle Informationen zur Verfügung stehen.

Prüfgrundlage:

bereitgestellte Artefakte des Standards

Prüfinhalt:

der Standard enthält alle erforderlichen Bestandteile

- XÖV-UML-Modell,
- XML-Schema-Dateien,
- Dokumentation (Datenstrukturen, eventuell Prozesse)

K-4 (MUSS): Veröffentlichung

Die Zertifizierung ist ausschließlich über das XRepository zu beantragen. Der Standard muss mit seiner Dokumentation als PDF-Datei, seinen XML-Schema-Dateien, einer XMI-Repräsentation seines XÖV-UML-Modells sowie seinem Pflegekonzept nach erfolgter Zertifizierung unverzüglich im XRepository veröffentlicht werden.

Begründung:

eine zentrale Anlaufstelle, einfache Auffindbarkeit

Prüfgrundlage:

Inhalte des XÖV-Standards im XRepository

Prüfinhalt:

Vorhandensein von XML-Schema-Dateien, Dokumentation als PDF, XÖV-UML-Modell als XMI-Datei sowie Pflegekonzept

K-5 (MUSS): Nachhaltigkeit des Standards

Für den XÖV-Standard muss ein Pflegekonzept vorliegen, aus dem erkennbar ist, dass eine langfristige Wartung und Fortschreibung gewährleistet wird.

Begründung:

Investitionssicherheit für implementierende Fachverfahrenshersteller sowie für Standards, die explizit auf anderen Standards aufbauen und mit diesen kommunizieren.

Prüfgrundlage:

Pflegekonzept

Prüfinhalt:

Vorhandensein eines Pflegekonzeptes mit Angaben zur für die Pflege zuständigen Stelle, Aufgaben, Rollen und Verantwortlichkeiten sowie zur Finanzierung

2.3.2 Auskunftspflichten der Standardentwickler und -betreiber

Auskunftspflichten beziehen sich auf Kriterien, bei denen die Verantwortlichen eines Standards Informationen zu ihrem Vorhaben aufbereiten und an die XÖV-Koordination übermitteln. Diese Informationen dienen im Wesentlichen der Transparenz zu Inhalten und Rahmenbedingungen des Standardisierungsvorhabens für die XÖV-Koordination, aber auch für Dritte, um die Wiederverwendung fachlicher Datenschnittstellen zu fördern.

K-6 (MUSS): Anzeige der Entwicklungsabsicht

Der Beginn der Entwicklung eines Standards muss der XÖV-Koordination so früh wie möglich nach der Identifizierung des Bedarfs angezeigt werden.

Begründung:

Redundante Entwicklungen für ein und denselben fachlichen Sachverhalt sollen vermieden und Synergien ermöglicht werden.

Prüfgrundlage:

initialer Steckbrief, Begründung eventueller Abweichung

Prüfinhalt:

Fachgebiet und Zweck des Standards, Begründung eventueller Abweichung

K-7 (MUSS): Informationen zum Status quo des Standards

Die für die Entwicklung und die Pflege des Standards zuständige Stelle (Organisationseinheit der öffentlichen Verwaltung) muss den Projektsteckbrief ausfüllen und an die XÖV-Koordination übermitteln. Bei relevanten Änderungen muss die zuständige Stelle den Steckbrief aktualisieren.

Begründung:

Schaffen von Transparenz hinsichtlich der XÖV-Standards. Redundante Entwicklungen für ein und denselben fachlichen Sachverhalt sollen vermieden und Synergien ermöglicht werden.

Prüfgrundlage:

Steckbrief

Prüfinhalt:

der Steckbrief ist aktuell und regelgerecht ausgefüllt

2.3.3 Technische Kriterien

Die technischen Kriterien der XÖV-Konformität beziehen sich auf das XÖV-UML-Modell (d. h. Prozesse und Datenstrukturen in UML 2.x Notation) und seine Darstellung in XML-Schema. Diese Kriterien sind weitestgehend automatisiert überprüfbar.

Die XÖV-konforme Generierung der XML-Schemata aus dem XÖV-UML-Modell und die automatisierte Überprüfung der technischen Kriterien sind in dem XÖV-Produktionszubehör – insbesondere im XGenerator und den XÖV-XSD-Vorlagen – implementiert, das in die Produktionsumgebung eines jeden XÖV-Vorhabens zur Erstellung eines XÖV-Standards integriert sein muss.

K-8 (SOLL): Modellierung der Prozesse in UML

Die verteilten Datenverarbeitungsprozesse, in denen die durch den XÖV-Standard spezifizierten Nachrichten ausgetauscht werden, sollen unter Verwendung von UML 2.x als Aktivitätsdiagramme beschrieben werden.

Begründung:

Das Verständnis der Prozesse ist wichtig für die Spezifikation konkreter Nachrichten. UML ist ein anerkannter Modellierungsstandard für Prozesse.

Prüfgrundlage:

XÖV-UML-Modell im elektronischen Format, Begründung eventueller Abweichung

Prüfinhalt:

UML-Aktivitätsdiagramme in Version 2.x, Begründung eventueller Abweichung

K-9 (MUSS): Modellierung der Datenstrukturen in UML

Die Modellierung der Datenstrukturen des XÖV-Standards muss unter Verwendung von UML 2.x als Modellierungssprache erfolgen.

Begründung:

Der anerkannte Modellierungsstandard UML bietet eine geeignete Abstraktion für die Beschreibung von Datenstrukturen und erlaubt eine integrierte Sicht auf die Prozesse und Strukturen eines Standards. Die Modellierung in UML ist eine Voraussetzung für die Verarbeitung durch das XÖV-Produktionszubehör. Das XÖV-UML-Modell ist Grundlage für die fachliche Prüfung.

Prüfgrundlage:

XÖV-UML-Modell im elektronischen Format

Prüfinhalt:

UML-Klassendiagramm in Version 2.x

K-10 (MUSS): Einhaltung der XÖV-Namens- und Entwurfsregeln

Für XÖV-Standards müssen die von der XÖV-Koordination herausgegebenen XÖV-Namens- und Entwurfsregeln entsprechend ihrer Verbindlichkeit verwendet werden. Das schließt die Verwendung des von der XÖV-Koordination veröffentlichten XÖV-Profils für UML in der zum Zeitpunkt der Konformitätsprüfung jeweils aktuellen Fassung ein.

Begründung:

Sicherstellung der Interoperabilität von XÖV-Standards bereits auf UML-Ebene; Das XÖV-Profil regelt die einheitliche Modellierung der Prozess- und Datenstrukturen in UML gemäß der XÖV-Namens- und Entwurfsregeln. Die Verwendung des XÖV-Profils ist eine Voraussetzung für die Verarbeitung durch das XÖV-Produktionszubehör – insbesondere durch den XGenerator und die XÖV-XSD-Vorlagen. (Die detaillierten Namens- und Entwurfsregeln sind in dem “XÖV-Namens- und Entwurfsregeln” auf Seite 61 aufgeführt).

Sicherstellung der Konsistenz zwischen UML- und XML-Datenstrukturen eines XÖV-Standards und der Interoperabilität auf XML-Schema-Ebene.

Prüfgrundlage:

XÖV-UML-Modell als XMI-Repräsentation in einer für den XGenerator lesbaren Form

Prüfinhalt:

Validierung des XÖV-UML-Modells und Generierung der XML-Schemata durch Einsatz des XGenerators

K-11 (SOLL): Nutzung von XÖV-Kern- und Fachkomponenten

Die durch die XÖV-Koordination im XRepository veröffentlichten XÖV-Kern- und Fachkomponenten sollen im XÖV-UML-Modell wiederverwendet werden.

Begründung:

Die häufige Wiederverwendung gleicher bzw. ähnlicher Dateninhalte- und strukturen verbessert die Interoperabilität von XÖV-Standards und erleichtert deren Implementierung.

Prüfgrundlage:

XÖV-UML-Modell im elektronischen Format, Begründung eventueller Abweichungen

Prüfinhalt:

Datenstrukturen im XÖV-UML-Modell hinsichtlich verwendeter XÖV-Fachkomponenten, Begründung eventueller Abweichungen

K-12 (SOLL): Nutzung der XÖV-Basisdatentypen

Die von der XÖV-Koordination herausgegebenen XÖV-Basisdatentypen sollen im XÖV-UML-Modell verwendet werden.

Begründung:

Für den Fall, dass die XÖV-Basisdatentypen für die Verwendung im Standard geeignet sind, sollen sie anstelle eigener Datentypen eingesetzt werden. Die häufige Wiederverwendung gleicher Datentypen verbessert die Interoperabilität und erleichtert die Implementierung.

Prüfgrundlage:

XÖV-UML-Modell im elektronischem Format, Begründung eventueller Abweichungen

Prüfinhalt:

Datenstrukturen im XÖV-UML-Modell hinsichtlich verwendeter XÖV-Basisdatentypen, Begründung eventueller Abweichungen

K-13 (SOLL): Nutzung von Codelisten

Die von der XÖV-Koordination empfohlenen und im XRepository bereitgestellten Codelisten sollen verwendet werden.

Begründung:

Verbesserung der Interoperabilität durch Verwendung einheitlicher Schlüssel (Codes).

Prüfgrundlage:

XÖV-UML-Modell im elektronischem Format, Begründung eventueller Abweichung

Prüfinhalt:

Datenstrukturen hinsichtlich der verwendeten Codelisten, Begründung eventueller Abweichung

K-14 (MUSS): Erfolgreiche Verarbeitung des XÖV-UML-Modells durch das XÖV-Produktionszubehör

Das XÖV-UML-Modell muss fehlerfrei durch das von der XÖV-Koordination herausgegebene XÖV-Produktionszubehör in der zum Zeitpunkt der Konformitätsprüfung jeweils aktuellen Fassung verarbeitet werden können. Dies beinhaltet die fehlerfreie Erzeugung der XML-Schemata.

Begründung:

Der noch herrschende Konflikt zwischen (automatisierter) Prüfbarkeit und Offenheit wurde hier zugunsten der Prüfbarkeit entschieden.

Prüfgrundlage:

XÖV-UML-Modell als XMI-Repräsentation in einer für den XGenerator lesbaren Version

Prüfinhalt:

Einhaltung der technisch implementierten Namens- und Entwurfsregeln

K-15 (SOLL): Nutzung einer sicheren Infrastruktur für den elektronischen Datenaustausch

Die öffentliche Verwaltung entwickelt und betreibt Infrastrukturkomponenten, die sich an sicheren elektronischen Diensten (Secure Web Services) orientieren. Neben der dafür erforderlichen Standardisierung elektronischer Dienste auf fachlicher Ebene ist vor allem auch die Sicherheit bei der Inanspruchnahme und Erbringung der Services zu gewährleisten. Methodische und technische Grundlagen der fachlichen Standardisierung und der Infrastrukturkomponenten sind aufeinander abgestimmt.

Die Wirtschaftlichkeit von Infrastrukturkomponenten ist umso höher, je größer die Zahl der Nutzer ist. Aus diesem Grunde, und wegen der abgestimmten Weiterentwicklung fachlicher und sicherheitstechnischer Standards im Sinne sicherer elektronischer Dienste, empfehlen die OSCI-Leitstelle Bremen und das Bundesministerium des Innern (BMI) die angemessene Nutzung der von der öffentlichen Verwaltung entwickelten Infrastrukturkomponenten.

Ein XÖV-Standard soll daher, zur Erfüllung der in dem jeweiligen fachlichen Kontext notwendigen Sicherheitsanforderungen, die von der öffentlichen Verwaltung entwickelten Lösungen in angemessenem Umfang berücksichtigen. Hierzu zählen:

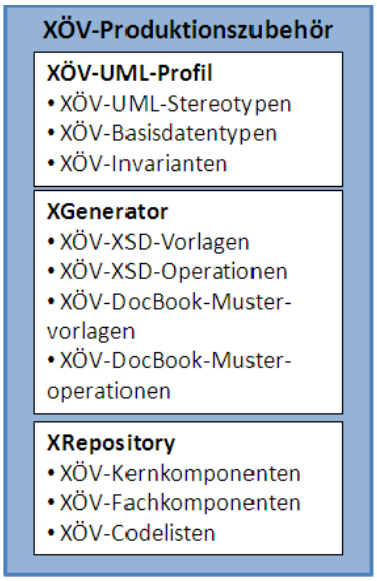
- Public-Key Infrastruktur PKI-1 Verwaltung,
- Übertragungsstandard OSCI-Transport und
- Service-Registry DVDV.

3. Produktion von XÖV-Standards

Dieses Kapitel gibt einen Überblick über den Prozess zur Erzeugung eines XÖV-Standards und die hierfür benötigten Mittel.

3.1 XÖV-Produktionszubehör der XÖV-Koordination

Die XÖV-Koordination gibt für die Erzeugung eines XÖV-Standards das XÖV-Produktionszubehör unter http://www.xoev.de heraus.



Das XÖV-Produktionszubehör umfasst Beschreibungen von technischen und semantischen Sachverhalten, XÖV-Werkzeuge sowie zu den Werkzeugen gehörende Dateien.

Hinweis: Einige Bestandteile des Produktionszubehörs müssen im Rahmen der XÖV-Konformität (siehe Abschnitt 2 auf Seite 13) zwingend durch die einzelnen XÖV-Vorhaben für die Erzeugung eines XÖV-Standards herangezogen werden, andere wiederum bilden ein zentrales Angebot, dessen Nutzung nicht vorgeschrieben ist.

Die nachfolgende Tabelle nennt die Bestandteile des XÖV-Produktionszubehörs, gibt Auskunft über deren Relevanz für die XÖV-Konformität und verweist auf die weiterführenden Abschnitte im XÖV-Handbuch.

Bestandteil	Kategorie	relevant für XÖV-Konformität	Beschreibung
XÖV-UML-Stereotypen	technische Beschreibung	ja	Teil des XÖV-UML-Profils, kennzeichnet Bestandteile im XÖV-UML-Modell bzgl. ihrer Verarbeitung durch den XGenerator (siehe Abschnitt 4.2 auf Seite 27)
XÖV-Basisdatentypen	technische Beschreibung und XML-Schema-Dateien	ja	Fachunabhängige Bausteine des XÖV-UML-Profils, die aus den W3C-Datentypen sowie XÖV-spezifischen Datentypen bestehen und als XML-Schema-Dateien zur Verfügung stehen (siehe Abschnitt 4.3 auf Seite 54 und Abschnitt 4.3 auf Seite 54)
XÖV-Invarianten	technische Beschreibung und Dateien für den XGenerator	ja[1]	Bestandteil des XÖV-UML-Profils, der Regeln zur Prüfung der korrekten Anwendung der XÖV-UML-Stereotypen und XÖV-Namens- und Entwurfsregeln beinhaltet und in Form von Dateien für den Einsatz im XGenerator zur Verfügung steht (siehe Abschnitt 5 auf Seite 61 und Abschnitt 9.1.2.1 auf Seite 134)
XGenerator	XÖV-Werkzeug	ja	XÖV-Werkzeug zur Erzeugung von XML-Schema-Dateien eines XÖV-Standards sowie optional zur Generierung von DocBook-XML-Dateien auf der Grundlage eines XÖV-UML-Modells (siehe Abschnitt 9 auf Seite 131)
XÖV-XSD-Vorlagen	Dateien für den XGenerator	ja	Dateien zur Erzeugung von XML-Schema-Dateien mittels XGenerators (siehe Abschnitt 9 auf Seite 131 und Abschnitt 9.1.2.2 auf Seite 135)
XÖV-XSD-Operationen	Dateien für den XGenerator	ja	Dateien zur Erzeugung von XML-Schema-Dateien mittels XGenerators (siehe Abschnitt 9 auf Seite 131 und Abschnitt 9.1.2.3 auf Seite 135)
XÖV-DocBook-Mustervorlagen	Dateien für den XGenerator	nein	Dateien zur Erzeugung von DocBook-XML-Dateien mittels XGenerators (siehe Abschnitt 9 auf Seite 131 und Abschnitt 9.1.2.2 auf Seite 135)
XÖV-DocBook-Musteroperationen	Dateien für den XGenerator	nein	Dateien zur Erzeugung von DocBook-XML-Dateien mittels XGenerators (siehe Abschnitt 9 auf Seite 131 und Abschnitt 9.1.2.3 auf Seite 135)
XRepository	XÖV-Werkzeug	ja	XÖV-Werkzeug zur Veröffentlichung von XÖV-Standards sowie zum Bezug von XÖV-Kernkomponenten, XÖV-Fachkomponenten und fachunabhängigen Codelisten (siehe Abschnitt 2 auf Seite 13)
XÖV-Kernkomponenten	semantische Beschreibungen	ja[2]	Fachübergreifende Bausteine zur Integration in einen XÖV-Standard (siehe Abschnitt 7 auf Seite 101)

Bestandteil	Kategorie	relevant für XÖV-Konformität	Beschreibung
XÖV-Fachkomponenten	semantische Beschreibungen	ja[3]	Fachspezifische Bausteine zur Integration in einen XÖV-Standard (in XÖV-Handbuch Version 1.1)
XÖV-Codelisten	semantische Beschreibungen	ja[4]	Allgemeine Codelisten zur Integration in einen XÖV-Standard (siehe Abschnitt 2 auf Seite 13)

1.Invarianten zu den MUSS-Regeln müssen immer, diejenigen zu den SOLL-Regeln sollten immer erfüllt sein – Invarianten zu Empfehlungen können optional eingesetzt werden (siehe Abschnitt 5 auf Seite 61)

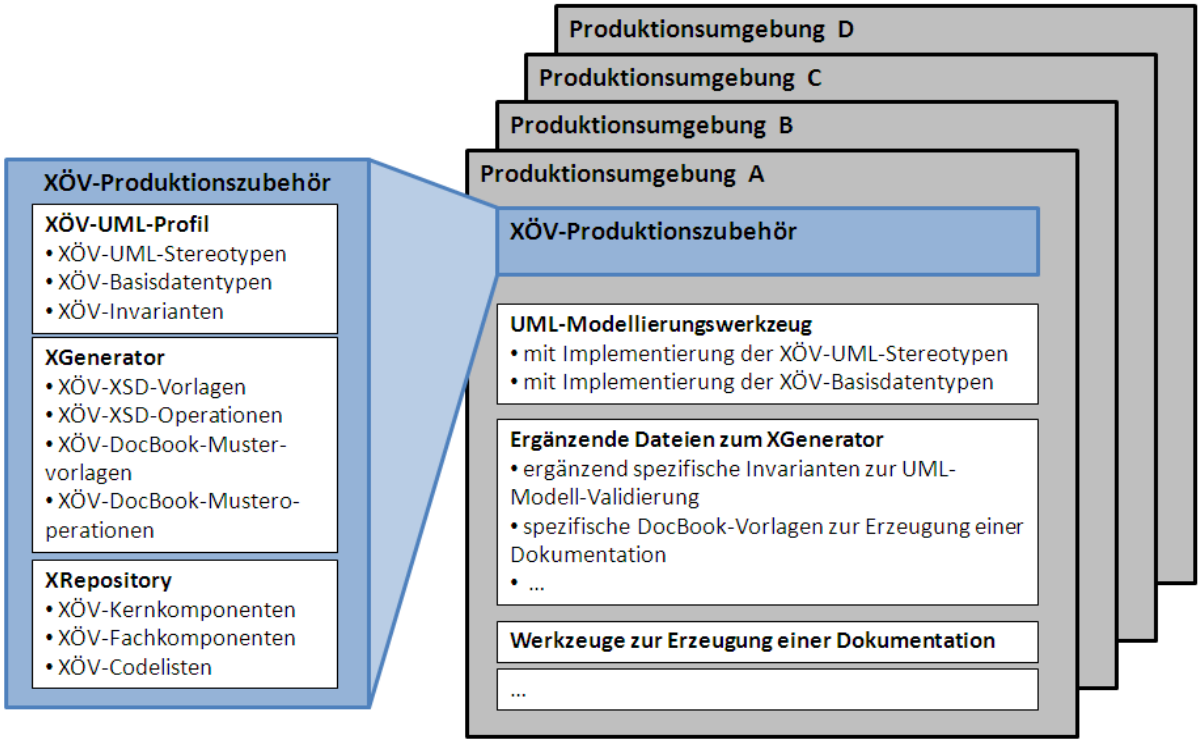
2.bei fachlicher Eignung

3.bei fachlicher Eignung

4.bei fachlicher Eignung

3.2 Produktionsumgebungen der XÖV-Vorhaben

Jedes XÖV-Vorhaben benötigt für die Erstellung eines XÖV-Standards eine Produktionsumgebung. Kern jeder Produktionsumgebung ist das XÖV-Produktionszubehör (siehe Abschnitt 3.1 auf Seite 21). Über das XÖV-Produktionszubehör hinaus muss jedes XÖV-Vorhaben bzw. eine Institution, die mehrere XÖV-Standards betreut, weitere, ergänzende Werkzeuge und Dateien spezifisch für den eigenen Bedarf in einer eigenen Produktionsumgebung zusammenfassen.



Ein Beispiel für einen spezifischen Bestandteil einer Produktionsumgebung ist ein UML-Modellierungswerkzeug mit einer konkreten Implementierung der XÖV-UML-Stereotypen sowie der XÖV-Basisdatentypen. Weiterhin erfolgt die bedarfsorientierte Anpassung der XÖV-DocBook-Mustervorlagen, die Erstellung neuer DocBook-Vorlagen oder auch die Verwendung anderer Lösungswege zur Erzeugung der Dokumentation in der für ein XÖV-Vorhaben definierten Produktionsumgebung.

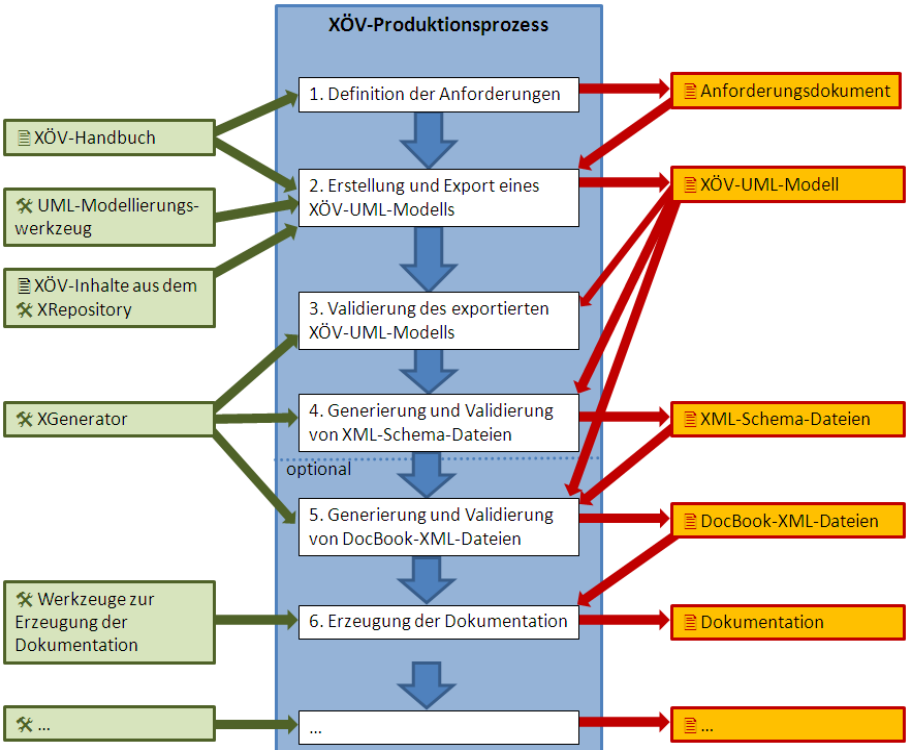
Hinweis: Die alleinige Verwendung des durch die XÖV-Koordination zur Verfügung gestellten XÖV-Produktionszubehörs ist für die Erstellung eines XÖV-Standards nicht ausreichend. Ein XÖV-Vorhaben muss selbst die Voraussetzungen für die Nutzung des XÖV-Produktionszubehörs in einer eigenen Produktionsumgebung schaffen.

Die XÖV-Koordination bietet unter http://www.xoev.de den XÖV-Vorhaben eine Plattform an, ihre Produktionsumgebung zur Verfügung zu stellen bzw. an Lösungen anderer XÖV-Vorhaben zu partizipieren.

3.3 XÖV-Produktionsprozess

Der XÖV-Produktionsprozess beschreibt die essentiellen Schritte, die von einem ein XÖV-Vorhaben für die Erstellung eines XÖV-Standards durchlaufen werden. Sie beziehen sich hierbei entweder auf die direkte Anwendung des XÖV-Produktionszubehörs (siehe Abschnitt 3.1 auf Seite 21) oder dessen Implementierung in der eigenen Produktionsumgebung (siehe Abschnitt 3.2 auf Seite 23).

Die nachfolgende Abbildung zeigt den Verlauf der essentiellen Schritte sowie die Voraussetzungen, Mittel und Ergebnisse der einzelnen Schritte.



Die Anforderungen an einen zukünftigen XÖV-Standard werden von Fachexperten definiert. Die nachfolgenden Schritte werden von XÖV-Modellierer (XPfleger) ausgeführt.

4. XÖV-UML-Profil

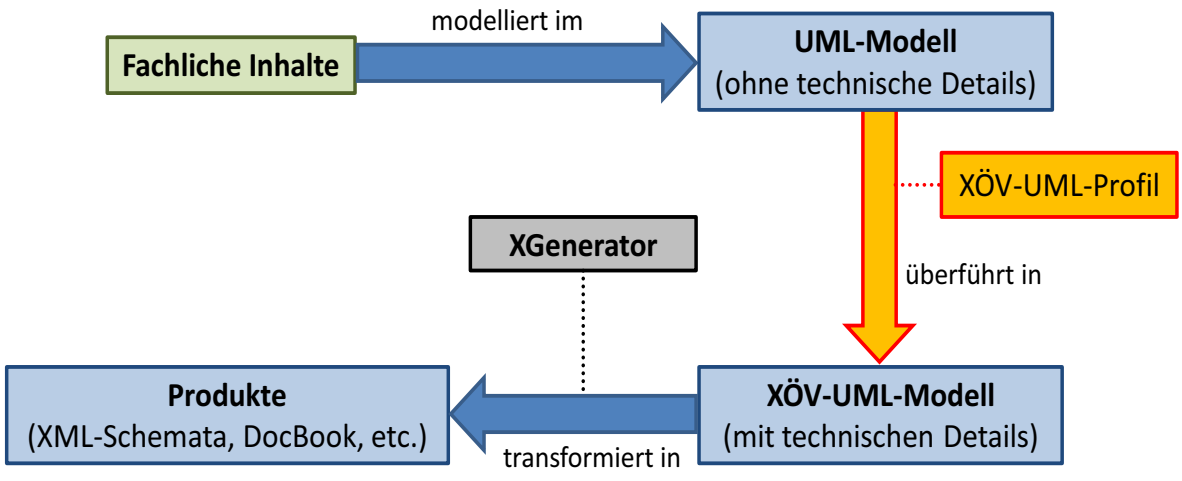
In der Entwicklung eines XÖV-Standards haben es die Verantwortlichen mit zwei verschiedenen Modellen der Fachdaten zu tun: einem konzeptionellen Modell, welches die Fachgruppen festlegen und einem physischen Modell für die elektronische Übertragung von XML-Dokumenten, beschrieben durch XML-Schemata.

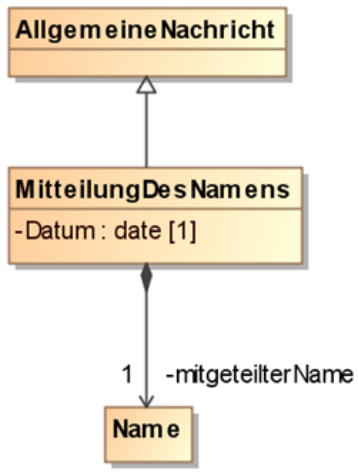
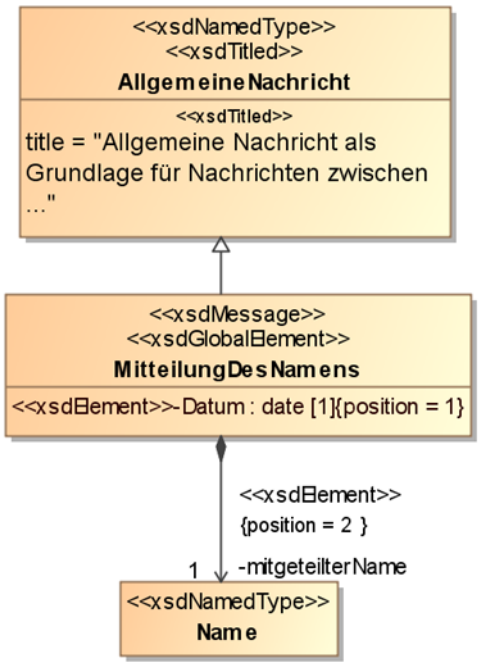
Das konzeptionelle Modell bildet die fachliche Welt möglichst direkt ab, ohne Überlegungen zur technischen Übertragung zu formulieren. Das Erstellen der XML-Schemata erfordert zusätzlichen Aufwand, um das physische Layout der Fachdaten in Form von XML-Dokumenten umzusetzen.

XÖV betrachtet das konzeptionelle Fachmodell in Form von UML-Diagrammen mit erläuterndem Text. Auf dieser Grundlage lässt sich die fachliche Modellierung und Abstimmung strukturiert und vereinfacht durchführen, ohne bereits auf der technischen Ebene die XML-Schemata zu betrachten.

Da das physische Modell in Form von XML-Schemata nur eine Verfeinerung des UML-Modells ist – beide Modelle beschreiben dieselben Daten –, bietet es sich an, die XML-Schemata automatisiert und einheitlich aus dem UML-Modell zu erzeugen. Allerdings werden für die XML-Schemata weitere Informationen benötigt, die im reinen Fachmodell so nicht vorliegen. Diese Informationen umfassen beispielsweise die Benennung von Namensräumen, unter welcher URL das Schema zu finden ist und die Abbildung auf die W3C-Schema-Basistypen. Diese Informationen sind rein technischer Natur und fügen dem Fachmodell konzeptionell keine neuen fachlichen Informationen hinzu.

Als Vorgehen für die Abbildung solcher Zusatzinformationen in UML wurde die Definition des XÖV-UML-Profils als Bestandteil des XÖV-Produktionszubehörs (siehe Abschnitt 3.1 auf Seite 21) erstellt.



Beispiel *ohne* technische Details	Beispiel *mit* technischen Details
	

UML-Profile werden genutzt, um die UML, die für viele verschiedene Zwecke genutzt werden kann domänenspezifisch zu machen. In diesem Fall ist die Domäne XÖV.

Das XÖV-UML-Profil versetzt den Modellierer in die Lage, UML-Klassendiagrammen systematisch Zusatzinformationen zum Erzeugen von XML-Schemata und Dokumentation hinzuzufügen. Für den Produktionsprozess eines XÖV-Standards bedeutet dies:

- Der Modellierer verwendet ein UML-Werkzeug, um die fachlichen Inhalte mit Hilfe von UML-Diagrammen anzulegen.
- Durch das XÖV-UML-Profil stehen technische Mittel zur Verfügung, mit denen XML-Eigenschaften definiert werden.
- Das mit dem XÖV-UML-Profil angereicherte UML-Modell (XÖV-UML-Modell) wird zur Grundlage für die Weiterverarbeitung durch den XGenerator (siehe Abschnitt 9 auf Seite 131). Dieser überprüft das UML-Modell auf die korrekte Anwendung des XÖV-UML-Profils sowie weiterer Regeln und erstellt dann automatisiert die XML-Schemata, ggf. eine Dokumentation sowie weitere Artefakte (z. B. WSDL-Dateien).

4.1 Allgemeiner Aufbau eines UML-Profils

UML-Profile[1] gehören zu den von der *Object Management Group* (OMG)[2] definierten Standardmechanismen, um die UML für spezielle Domänen anzupassen und zu erweitern. Ein UML-Profil beinhaltet im Wesentlichen drei Bestandteile:

1. **Stereotypen und ihre Eigenschaften (TaggedValue):** Stereotypen erlauben es, UML-Konzepten (z. B. Modell, Paket, Klasse, Attribut) weitere Klassifikationen zuzuordnen. Stereotyp-Eigenschaften erlauben, der zusätzlichen Klassifikation weitere Details hinzuzufügen.

   Die Stereotypen des XÖV-UML-Profils werden in Abschnitt 4.2 auf Seite 27 beschrieben.

2. **Datentypen:** Die Datentypen können bestimmten UML-Elementen (z. B. UML-Klassenattributen) zugeordnet werden und geben somit Auskunft über deren Wertebereich auf XML-Schema-Ebene.

   Die Datentypen des XÖV-UML-Profils werden als UML-Klassen definiert, z. B. `Code` (siehe Abschnitt 4.3 auf Seite 54).

3. **Profilspezifische Einschränkungen (Invarianten):** In einem Profil können spezifische Regeln für die profilkonforme Verwendung der UML-Sprachelemente, der Stereotypen und der Eigenschaften sowie der Datentypen definiert werden. Im hier dargestellten XÖV-UML-Profil wird z. B. gefordert, dass beim Modellieren einer «xsdRestriction»-Beziehung dem spezielleren Typ keine neuen Attribute mehr hinzugefügt werden dürfen und dass Mehrfachvererbung für XML-Schema-Typen verboten ist, da dies im XML-Schema nicht realisierbar ist.

   Die Invarianten zu einem Profil werden auf technischer Ebene mit Hilfe der Sprache *Object Constraint Language* (OCL) definiert und sind werkzeugunterstützt (automatisiert) überprüfbar (siehe Abschnitt 9.1.2.1 auf Seite 134 und Abschnitt B auf Seite 157). Neben der Generierung von XML-Schemata und der Dokumentation ist diese Überprüfung eine wichtige Funktion des XGenerators (siehe Abschnitt 9 auf Seite 131).

   Die Invarianten zum XÖV-UML-Profil sind als technische Entsprechungen den Namens- und Entwurfsregeln in diesem Handbuch zugeordnet (siehe Abschnitt 5 auf Seite 61).

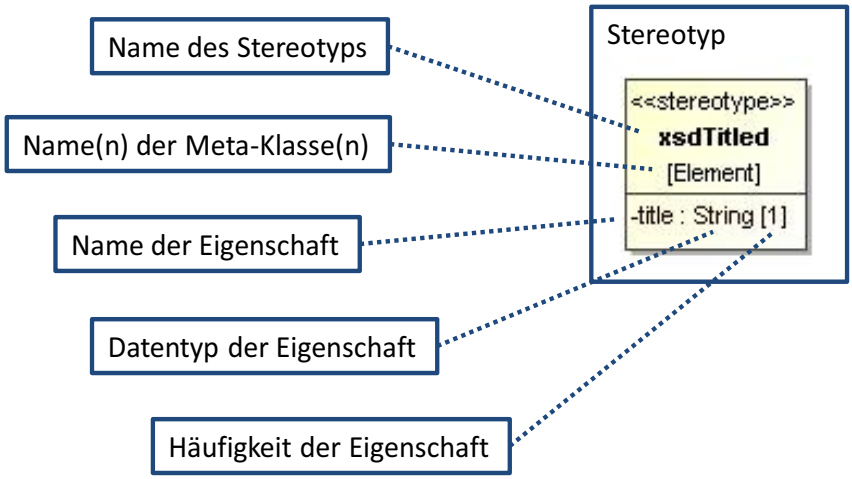
4.2 XÖV-Stereotypen

In diesem Abschnitt werden die UML-Stereotypen des XÖV-UML-Profils und ihre Eigenschaften erläutert.

Jeder Stereotyp besitzt einen Namen, der in doppelte französische Anführungszeichen («») gesetzt wird (siehe nachfolgende Abbildung). Weiterhin wird der Name des UML-Konzepts angegeben, das der Stereotyp zusätzlich klassifiziert. Als zusätzliche Informationen werden zu einem Stereotyp Eigenschaften definiert, die es erlauben, der zusätzlichen Klassifikation weitere Details hinzuzufügen. Eine Eigenschaft wird dabei mit einem Namen, einem Datentyp zur Einschränkung seiner möglichen Inhalte sowie der Häufigkeit angegeben.

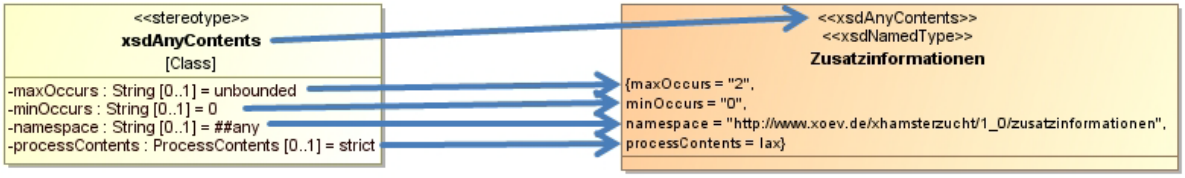
1.siehe http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML

2.siehe http://www.omg.org/



In den nachfolgenden Abschnitten werden die Stereotypen und ihre Eigenschaften vorgestellt. Es wird dabei immer mindestens die Abbildung des Stereotyps selbst dargestellt sowie gegebenenfalls ein Anwendungsbeispiel auf ein entsprechendes UML-Konzept (z.B. eine UML-Kasse). Die Stereotypen und deren Eigenschaften sind mit den Entsprechungen im UML-Konzept über Pfeilen verbunden.

4.2.1 xsdAnyContents



Anwendbar für: Class (UML-Klassen)

Eine mit diesem Stereotyp versehene UML-Klasse wird im XML-Schema als Typ umgesetzt, der beliebige XML-Elemente enthalten kann.

Die Umsetzung des Stereotyps wird im XML-Schema durch xs:any[1] realisiert.

Eigenschaften von xsdAnyContents		
Eigenschaft	**Typ**	**Häufigkeit**
namespace	String	0..1
processContents	ProcessContents	0..1
minOccurs	String	0..1
maxOccurs	String	0..1

1.siehe http://www.w3.org/TR/xmlschema-1/#Wildcards

4.2.1.1 namespace (String)

Die Eigenschaft legt den XML-Namensraum fest, aus dem die Inhalte kommen dürfen. Standardwert ist der Namensraum *“##any”*.

Der Wert wird direkt in das XML-Schema übernommen.

4.2.1.2 processContents (ProcessContents)

Mit dieser Eigenschaft kann der Wert des Attributs processContents definiert werden, der standardmäßig auf *“strict”* (Namensraum muss verfügbar sein und Validierung findet statt) eingestellt ist. Mögliche Werte sind ausserdem *“lax”* (Validierung nur wenn der Namensraum verfügbar ist) und *“skip”* (keine Validierung).

Der Wert wird direkt in das XML-Schema übernommen.

4.2.1.3 maxOccurs (String)

Mit dieser Eigenschaft wird die maximale Anzahl der Elemente aus dem in der Eigenschaft "namespace" definierten Namensraum festgelegt.

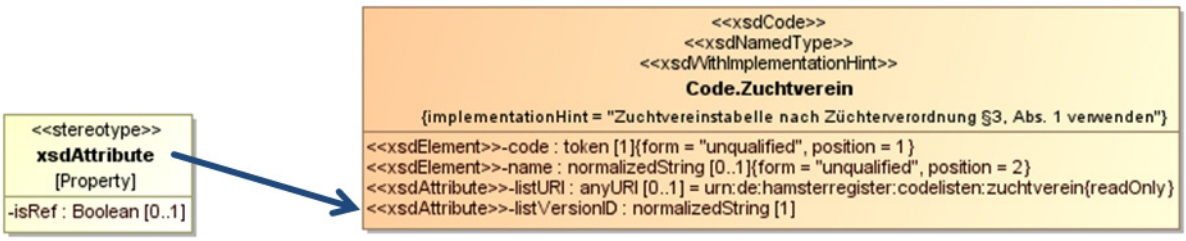
Der Wert wird direkt in das XML-Schema übernommen.

4.2.1.4 minOccurs (String)

Mit dieser Eigenschaft wird die minimale Anzahl der Elemente aus dem in der Eigenschaft "namespace" definierten Namensraum festgelegt.

Der Wert wird direkt in das XML-Schema übernommen.

4.2.2 xsdAttribute



Anwendbar für: Property (UML-Attribute und -Assoziationsenden)

Eine mit diesem Stereotyp gekennzeichnete UML-Eigenschaft (UML-Attribut oder -Assozationsende) wird als xs:attribute[1] umgesetzt. UML-Klassenattribute, die mit diesem Stereotyp gekennzeichnet sind, dürfen als Multiplizität nur 1 (dies ist der Standard-Wert) oder 0..1 besitzen, d.h. sie sind somit entweder obligatorisch oder optional.

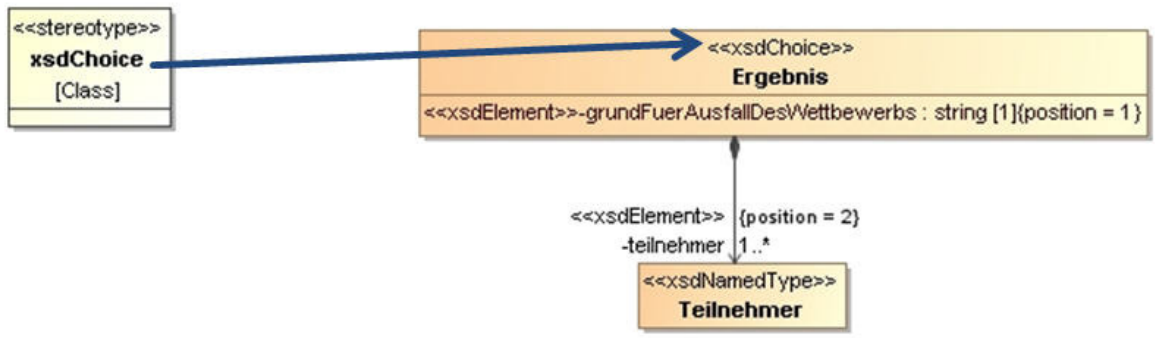
Eigenschaften von xsdAttribute		
Eigenschaft	**Typ**	**Häufigkeit**
isRef	Boolean	0..1

1.siehe http://www.w3.org/TR/xmlschema-1/#cAttribute_Declarations

4.2.2.1 isRef (Boolean)

Falls die als xsdAttribute markierte Eigenschaft auf eine globale Attributgruppe verweisen soll, muss für diese Eigenschaft der Wert true angegeben werden. Andernfalls wird ein lokales Attribut erzeugt.

4.2.3 xsdChoice

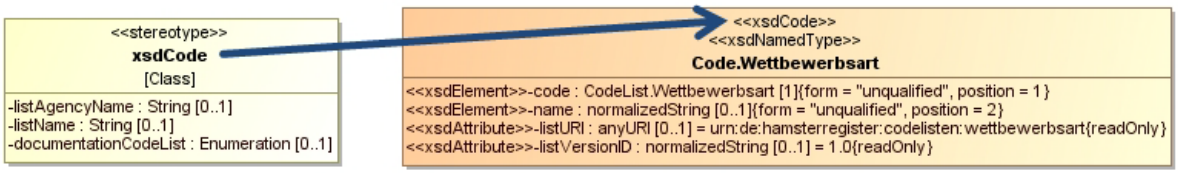


Anwendbar für: Class (UML-Klassen)

Eine UML-Klasse, die mit diesem Stereotyp versehen ist, wird im XML-Schema durch ein xs:choice[1]-Elementmodell umgesetzt (anstelle von xs:sequence[2]). Der "choice" erstreckt sich über alle Eigenschaften der mit "xsdChoice" gekennzeichneten UML-Klasse, die mit "xsdElement" gekennzeichnet sind.

Zu dem Stereotypen sind keine Eigenschaften definiert.

4.2.4 xsdCode



Anwendbar für: Class (UML-Klassen)

Eine UML-Klasse, die mit dem Stereotypen "xsdCode" versehen ist, stellt einen Code-Datentyp dar, der eine einzusetzende Codeliste näher beschreibt.

Die Umsetzung der so stereotypisierten UML-Klasse im XML-Schema erfolgt als xs:complexType[3] mit xs:complexContent[4] im XML-Schema.

1.siehe http://www.w3.org/TR/xmlschema-1/#Model_Groups

2.siehe http://www.w3.org/TR/xmlschema-1/#Model_Groups

3.siehe http://www.w3.org/TR/xmlschema-1/#Complex_Type_Definitions

4.siehe http://www.w3.org/TR/xmlschema-1/#declare-type

In Abhängigkeit von der zusätzlichen Vergabe des Stereotypen "xsdNamedType" wird der Code-Datentyp als benannter Typ im XML-Schema umgesetzt, ansonsten als anonymer Typ.

Eigenschaften von xsdCode		
Eigenschaft	**Typ**	**Häufigkeit**
listAgencyName	String	0..1
listName	String	0..1
documentationCodeList	Enumeration	0..1

4.2.4.1 listAgencyName (String)

Der Herausgeber der Codeliste.

Die Information dient nur zu Dokumentationszwecken und hat keine Auswirkung auf die Struktur des XML-Schemas: Es erfolgt eine Ausgabe in der Dokumentation des Code-Datentyps im XML-Schema.

4.2.4.2 listName (String)

Der offizielle Name der Codeliste.

Die Information dient nur zu Dokumentationszwecken und hat keine Auswirkung auf die Struktur des XML-Schema: Es erfolgt eine Ausgabe in der Dokumentation des Code-Datentyps im XML-Schema.

4.2.4.3 documentationCodeList (Enumeration)

Für Codelisten vom Typ 2 (nicht schemavalidierend) kann über diese Eigenschaft eine Verbindung zu einer Codeliste hergestellt werden. Dadurch kann bei der Dokumentationsgenerierung trotzdem auf eine Codeliste verwiesen werden.

4.2.5 xsdCodeList

Anwendbar für: Enumeration (UML-Aufzählungen)

Der Stereotyp ermöglicht die Kennzeichnung einer UML-Enumeration als eine Liste von Codes.

Die Umsetzung des Stereotyps im XML-Schema erfolgt als xs:simpleType[1] mit dem durch xs:enumeration[2] eingeschränkten Basistyp "xs:token"[3], es sei denn die Eigenschaft *base* definiert einen anderen Basistypen. Wird die Codeliste in einem Code Datentyp verwendet, muss der Basistyp vom Typ xs:token oder einer Unterklasse sein.

1.siehe http://www.w3.org/TR/xmlschema-1/#Simple_Type_Definitions

2.siehe http://www.w3.org/TR/xmlschema-2/#rf-enumeration

3.siehe http://www.w3.org/TR/xmlschema-2/#token

In Abhängigkeit von der zusätzlichen Vergabe des Stereotypen "xsdNamedType" wird die Codeliste als benannter Typ im XML-Schema umgesetzt, ansonsten als anonymer Typ.

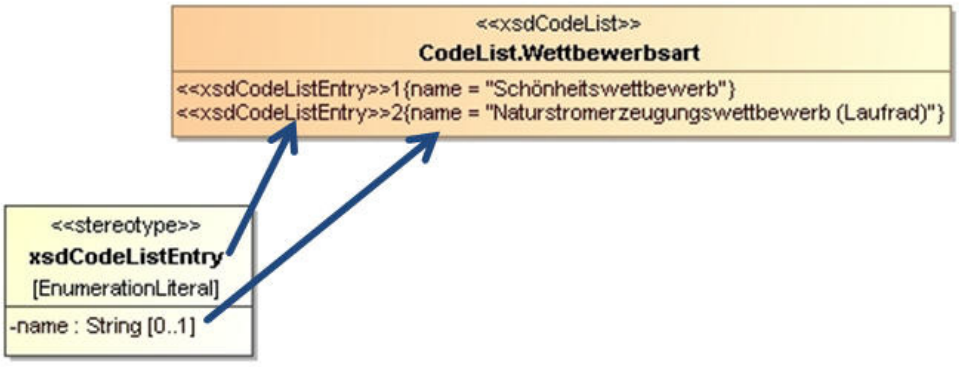
Eigenschaften von xsdCodeList		
Eigenschaft	**Typ**	**Häufigkeit**
base	Class	0..1

4.2.5.1 base (Class)

Bestimmt den Obertypen der Enumeration. Dieser muss ein einfacher Typ sein. Bei der Verwendung innerhalb eines XÖV-Codes muss er zusätzlich vom Typ xs:token order ein davon abgeleiteter Typ sein.

Der Wert wird direkt in das XML-Schema übernommen.

4.2.6 xsdCodeListEntry



Anwendbar für: EnumerationLiteral (Einträge von UML-Aufzählungen)

Mit diesem Stereotyp werden Literale einer UML-Enumeration als Wert einer Codeliste (UML-Enumeration mit dem Stereotyp "xsdCodeList") gekennzeichnet.

Die Umsetzung des Stereotyps im XML-Schema erfolgt als xs:enumeration[1].

Eigenschaften von xsdCodeListEntry		
Eigenschaft	**Typ**	**Häufigkeit**
name	String	0 .. 1
obsolete	Boolean	0 .. 1

4.2.6.1 name (String)

Der ausgeschriebene Name für einen Code.

1.siehe http://www.w3.org/TR/xmlschema-2/#rf-enumeration

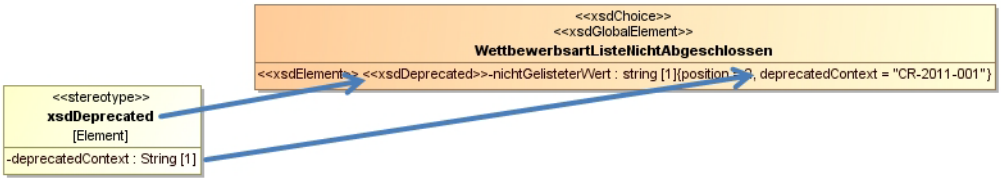
Die Information dient nur zu Dokumentationszwecken und hat keine Auswirkung auf die Struktur des XML-Schemas: Es erfolgt eine Ausgabe in der Dokumentation der jeweiligen xs:enumeration[1] im XML-Schema.

4.2.6.2 obsolete (Boolean)

Mit diesem Schalter können veraltete, nicht mehr zu übermittelnde Codes gekennzeichnet werden (obsolete = true). Deartig gekennzeichnete Codes, d. h. Literale einer Enumeration, werden im Kontext einer Standard-Codeliste nicht mehr in das Schema herausgeschrieben. Auf diese Weise kann die (unbemerkte) Wiederverwendung alter Codes für neue Zwecke, d. h. mit neuer Bedeutung, vermieden werden, da doppelte Codes in einer Enumeration in UML nicht zugelassen werden.

Der Default-Wert der Eigenschaft "obsolete" ist "false".

4.2.7 xsdDeprecated



Anwendbar für: Element (alle UML-Elemente)

Dieser Stereotyp kann bei der Dokumentationsgenerierung berücksichtigt werden, um bestimmte Teile eines Standards abzukündigen.

Der Stereotyp hat keine Auswirkungen auf die generierten XML-Schemata.

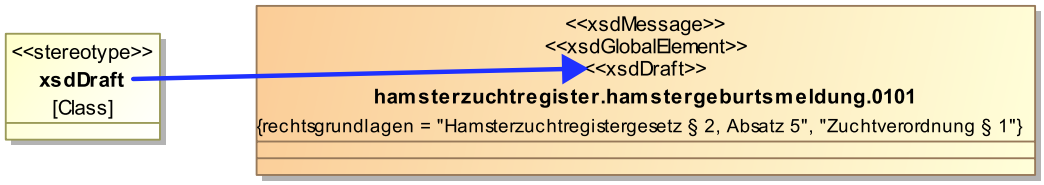
Eigenschaften von xsdDeprecated		
Eigenschaft	**Typ**	**Häufigkeit**
deprecatedContext	String	1

4.2.7.1 deprecatedContext (String)

Diese Eigenschaft wird dazu genutzt, um mehrere abgekündigte Bestandteile zu gruppieren, da diese z. B. durch einen einzigen Änderungeantrag wegfallen. Innerhalb der generierten Dokumentation kann dann auf einen beschreibenden Abschnitt verwiesen werden.

1.siehe http://www.w3.org/TR/xmlschema-2/#rf-enumeration

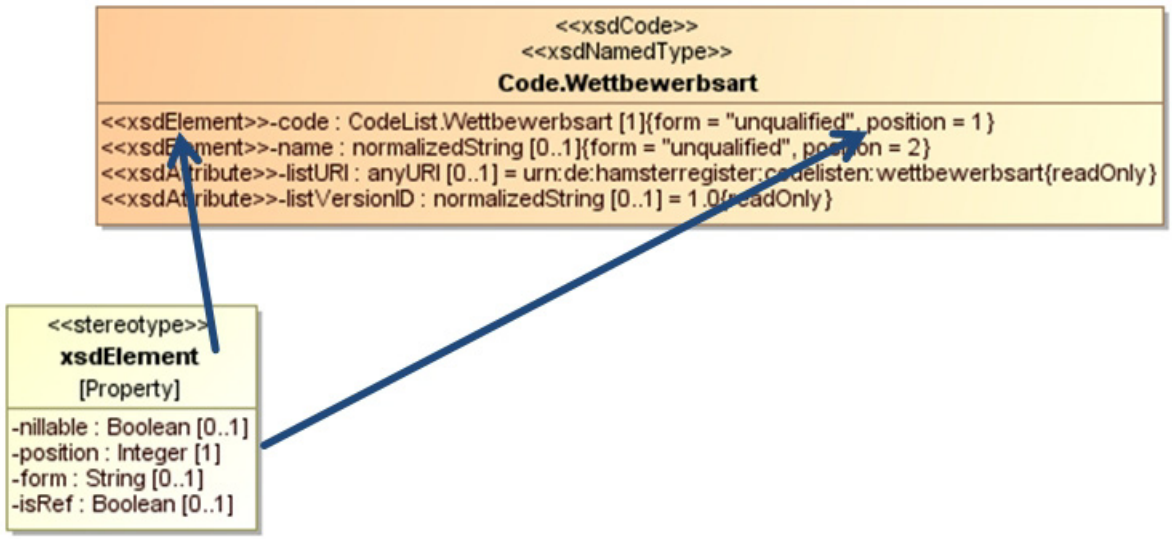
4.2.8 xsdDraft



Anwendbar für: Class (alle UML-Klassen)

Mit dem Stereotyp "xsdDraft" versehene UML-Klassen werden nicht in die Schema- und ggf. DocBook-Generierung einbezogen. Auf diese Weise können Inhalte (d. h. benannte Typen, Element- und Attributgruppen und globale Elemente) , die bereits mit XÖV-Stereotypen versehen sind und damit in die Generierung einbezogen würden, auf einfache Weise "auskommentiert" werden. Auskommentierte Inhalte dürfen nicht von "aktiven" Inhalten genutzt werden.

4.2.9 xsdElement



Anwendbar für: Property (UML-Attribute und -Assoziationsenden)

Eine mit diesem Stereotyp gekennzeichnete UML-Eigenschaft (UML-Attribut oder -Assoziationsende einer Aggregation) legt fest, dass sie als xs:element[1] umgesetzt wird.

Eigenschaften von xsdElement		
Eigenschaft	**Typ**	**Häufigkeit**
nillable	Boolean	0..1
position	Integer	1
form	String	0..1

1.siehe http://www.w3.org/TR/xmlschema-1/#cElement_Declarations

Eigenschaften von xsdElement		
Eigenschaft	**Typ**	**Häufigkeit**
isRef	Boolean	0..1

4.2.9.1 nillable (Boolean)

Wenn diese Eigenschaft mit "true" angegeben wird, wird dieses Element im entsprechenden xs:complexType[1] als "nillable='true'"[2] deklariert.

4.2.9.2 position (Integer)

Die Eigenschaft bestimmt die Position einer als xs:element[3] umgesetzten UML-Eigenschaft innerhalb der Elementsequenz xs:sequence[4] im entsprechenden xs:complexType.

Die Nummierung der Eigenschaft position muss für alle "xsdElement"-Eigenschaften einer UML-Klasse eindeutig und als positiver ganzzahliger Wert hinterlegt sein.

Die Nummern werden nicht in der Dokumentation des jeweiligen xs:element ausgegeben. Sie dient zur Bildung der xs:element-Reihenfolge.

4.2.9.3 form (String)

Standardgemäß werden xs:element mit form='qualified' herausgeschrieben. Mit Hilfe dieser Eigenschaft kann die Form für das aktuell betrachtete Element explizit bestimmt werden. Die Angabe des Wertes 'unqualified' ist für die Ableitung über unterschiedliche Namensräume notwendig.

Es können nur die Werte "unqualified" oder "qualified" angegeben werden.

4.2.9.4 isRef (Boolean)

Falls die als xsdElement markierte Eigenschaft auf ein globales Element oder eine Elementgruppe verweisen soll, muss für diese Eigenschaft der Wert true angegeben werden. Andernfalls wird ein lokales Element erzeugt.

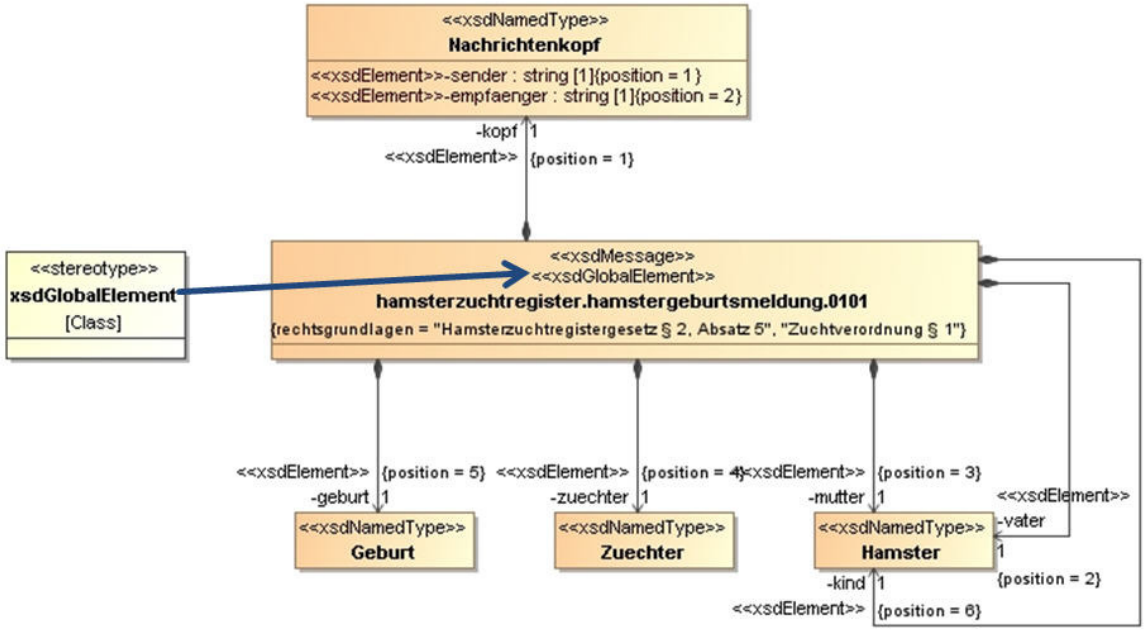
1.siehe http://www.w3.org/TR/xmlschema-1/#Complex_Type_Definitions

2.siehe http://www.w3.org/TR/xmlschema-1/#xsi_nil

3.siehe http://www.w3.org/TR/xmlschema-1/#cElement_Declarations

4.siehe http://www.w3.org/TR/xmlschema-1/#Model_Groups

4.2.10 xsdGlobalElement



Anwendbar für: Class (UML-Klassen)

Eine UML-Klasse, die mit dem Stereotyp "xsdGlobalElement" versehen ist, wird im XML-Schema als globales Element, d.h. als mögliches Wurzelelement-xs:element[1] ('document root') umgesetzt.

Der Stereotyp kann in Zusammenspiel mit dem Stereotypen "xsdMessage" zur Kennzeichnung von XÖV-Nachrichten verwendet werden.

Zu dem Stereotypen sind keine Eigenschaften definiert.

4.2.11 xsdGlobalElementType

Anwendbar für: Dependency (UML-Abhängigkeit)

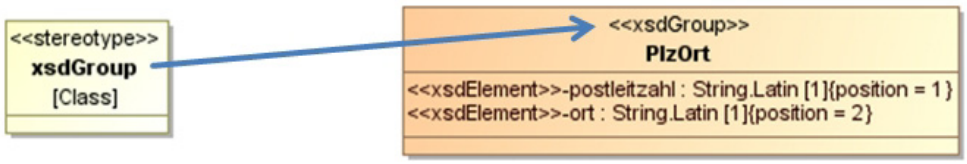
1.siehe http://www.w3.org/TR/xmlschema-1/#cElement_Declarations

Eine UML-Abhängigkeit zwischen einer als *xsdGlobalElement* ausgezeichneten UML-Klasse und einer als *xsdNamedType* ausgezeichneten UML-Klasse, die mit dem Stereotyp *xsdGlobalElementType* versehen ist, bewirkt, dass im XML-Schema als Datentyp des globalen Elements der benannte Typ, der als Ziel der Abhängigkeit angegeben ist, festgelegt wird[1].

Der Stereotyp muss im Zusammenspiel mit den Stereotypen *xsdGlobalElement* und *xsdNamedType* verwendet werden.

Zu dem Stereotypen sind keine Eigenschaften definiert.

4.2.12 xsdGroup

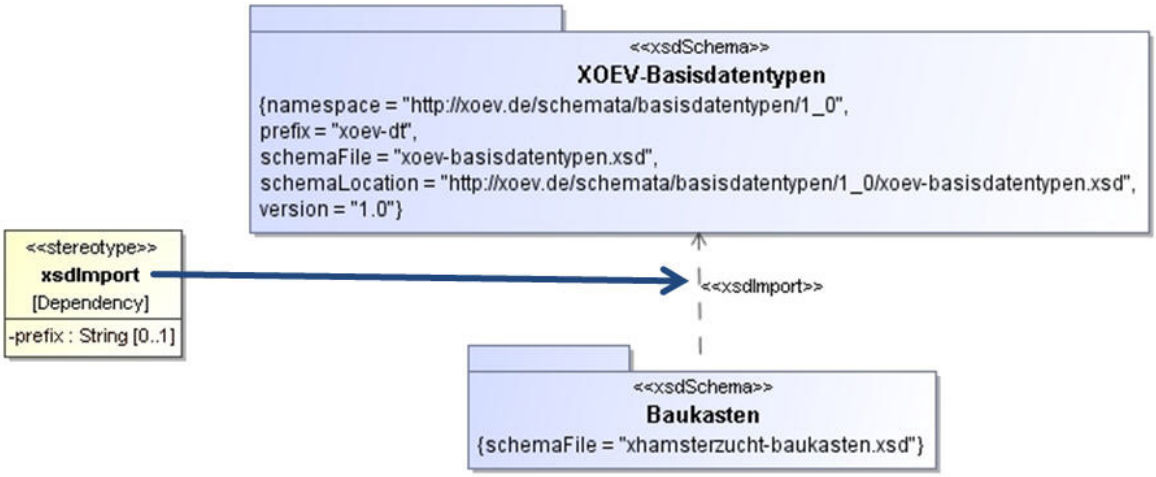


Anwendbar für: Class (UML-Klassen)

Eine UML-Klasse, die mit diesem Stereotyp versehen ist, wird im XML-Schema durch ein xs:group[2]- oder xs:attributeGroup[3]-Elementmodell umgesetzt (anstelle von xs:complexType). Die Gruppe enthält alle Eigenschaften der mit *xsdGroup* gekennzeichneten Klasse, die mit *xsdElement* bzw. *xsdAttribute* gekennzeichnet sind.

Zu dem Stereotypen sind keine Eigenschaften definiert.

4.2.13 xsdImport



Anwendbar für: Dependency (UML-Abhängigkeitsbeziehungen)

1.siehe http://www.w3.org/TR/xmlschema-1/#cElement_Declarations

2.siehe http://www.w3.org/TR/xmlschema-1/#cModel_Group_Definitions

3.siehe http://www.w3.org/TR/xmlschema-1/#cAttribute_Group_Definitions

Die mit dem Stereotyp gekennzeichnete UML-Abhängigkeit bildet eine Import-Beziehung zwischen zwei "xsdSchema"-UML-Paketen ab.

Das XML-Schema des Anbieter-Pakets (Ende mit Pfeilspitze) wird durch ein xs:import[1] in das XML-Schema des abhängigen Pakets eingebunden.

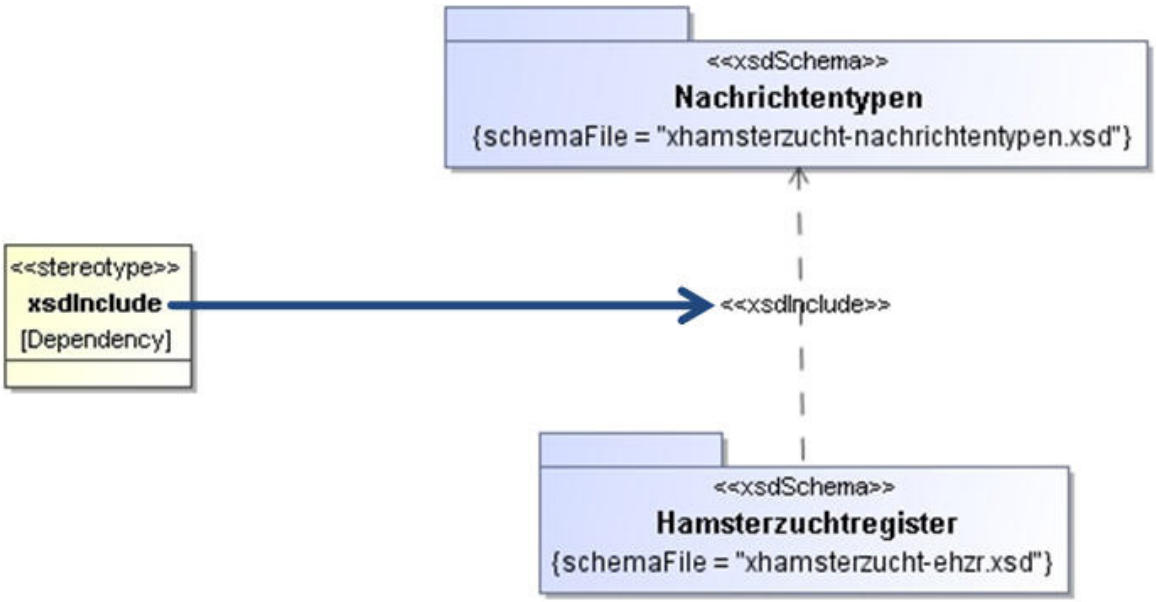
Eigenschaften von xsdImport		
Eigenschaft	**Typ**	**Häufigkeit**
prefix	String	0 .. 1

4.2.13.1 prefix (String)

Das XML-Präfix, mit dem der Namensraum des importierten Schemas angesprochen werden soll.

Der Wert wird direkt in das XML-Schema übernommen.

4.2.14 xsdInclude



Anwendbar für: Dependency (UML-Abhängigkeitsbeziehungen)

Die mit dem Stereotyp gekennzeichnete UML-Abhängigkeit bildet eine Beziehung zwischen zwei "xsd-Schema"-Paketen ab, wobei das eine Paket von dem anderen eingebunden wird.

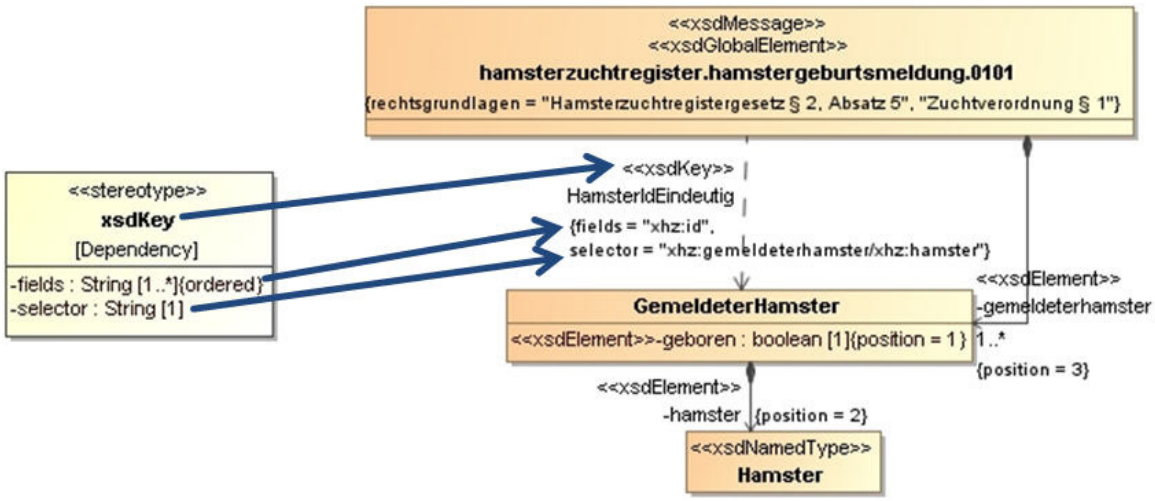
Das XML-Schema des Anbieter-Pakets wird durch ein xs:include[2] in das XML-Schema des abhängigen Pakets eingebunden.

Zu dem Stereotypen sind keine Eigenschaften definiert.

1.siehe http://www.w3.org/TR/xmlschema-1/#composition-schemaImport

2.siehe http://www.w3.org/TR/xmlschema-1/#compound-schema

4.2.15 xsdKey



Anwendbar für: Dependency (UML-Abhängigkeitsbeziehungen)

Eine mit diesem Stereotyp gekennzeichnete UML-Abhängigkeit modelliert ein xs:key-Constraint[1] im XML-Schema. Der Name der UML-Abhängigkeit wird als Name des xs:key[2] verwendet.

Eigenschaften von xsdKey		
Eigenschaft	**Typ**	**Häufigkeit**
fields	String	1 .. n
selector	String	1

4.2.15.1 fields (String)

Die Pfadausdrücke für die xs:field-Elemente[3].

4.2.15.2 selector (String)

Der Pfadausdruck für das xs:selector-Element[4].

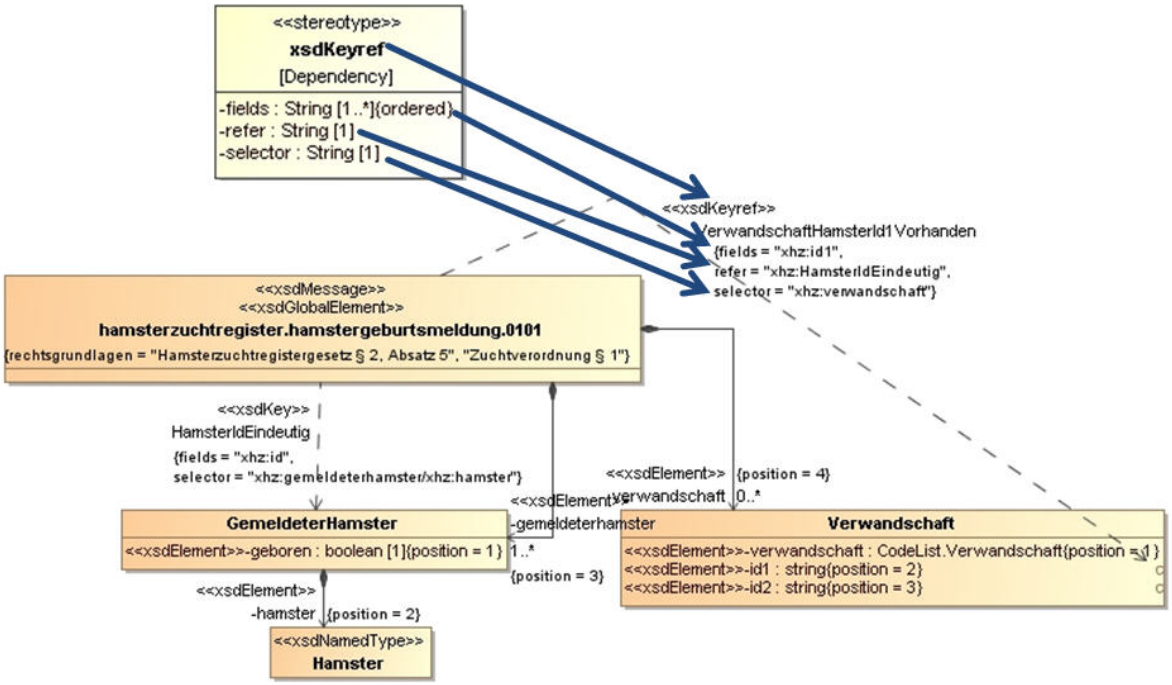
1.siehe http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions

2.siehe http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions

3.siehe http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions

4.siehe http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions

4.2.16 xsdKeyref



Anwendbar für: Dependency (UML-Abhängigkeitsbeziehungen)

Die mit diesem Stereotyp gekennzeichnete UML-Abhängigkeit modelliert ein xs:keyref[1]-Constraint in XML. Der Name der UML-Abhängigkeit wird als Name des xs:keyref[2] verwendet.

Da sich die Key-Constraints in XML-Schema immer auf ein XML-Element (und nicht auf einen XML-Typen) beziehen, werden sie auch in UML entsprechend an eine mit "xsdElement" versehene Klasseneigenschaft (UML-Attribut oder -Kompositition) annotiert.

Eigenschaften von xsdKeyref		
Eigenschaft	**Typ**	**Häufigkeit**
fields	String	1 .. n
refer	String	1
selector	String	1

4.2.16.1 fields (String)

Die Pfadausdrücke für die xs:field-Elemente[3].

1.siehe http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions

2.siehe http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions

3.siehe http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions

4.2.16.2 refer (String)

Der Name des xs:key- oder xs:unique-Constraints[1], auf das sich dieses xs:keyref-Constraint[2] bezieht.

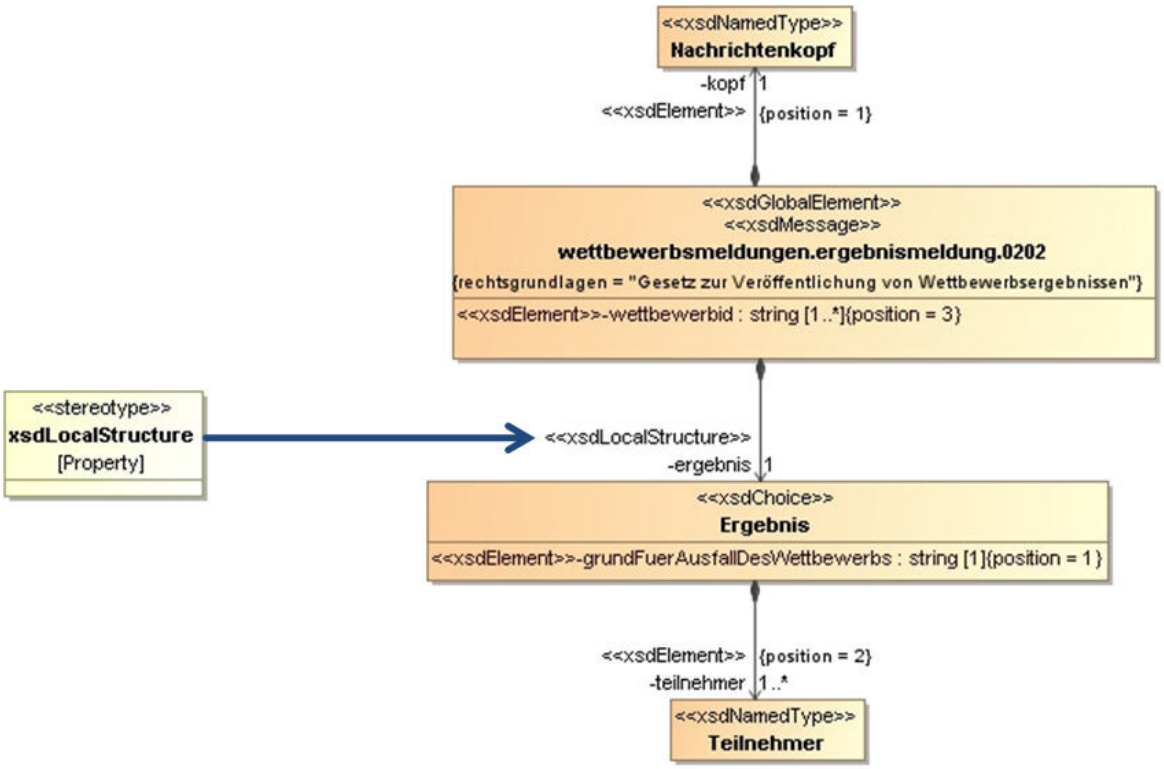
Die Umsetzung erfolgt als Attribut "refer" des xs:keyref-Elements[3].

4.2.16.3 selector (String)

Der Pfadausdruck für das 'refer'-Attribut im xs:keyref[4]. Es muss ein xs:key- oder xs:unique-Constraint[5] mit diesem Namen im Modell geben.

Die Umsetzung erfolgt als xs:selector[6].

4.2.17 xsdLocalStructure



Anwendbar für: Property (UML-Attribute und -Assoziationsenden)

Eine UML-Eigenschaft, die mit "xsdLocalStructure" versehen ist, kennzeichnet, dass die Strukturen der anonymen referenzierten UML-Klasse, die einen Typ mit komplexen Inhalt darstellt, im XML-Schema direkt, ohne Angabe von "xs:complexType"[7] in die darüberliegende Struktur eingebunden werden.

1.siehe http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions
2.siehe http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions
3.siehe http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions
4.siehe http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions
5.siehe http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions
6.siehe http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions
7.siehe http://www.w3.org/TR/xmlschema-1/#Complex_Type_Definitions

Verwendung ohne "xsdLocalStructure":

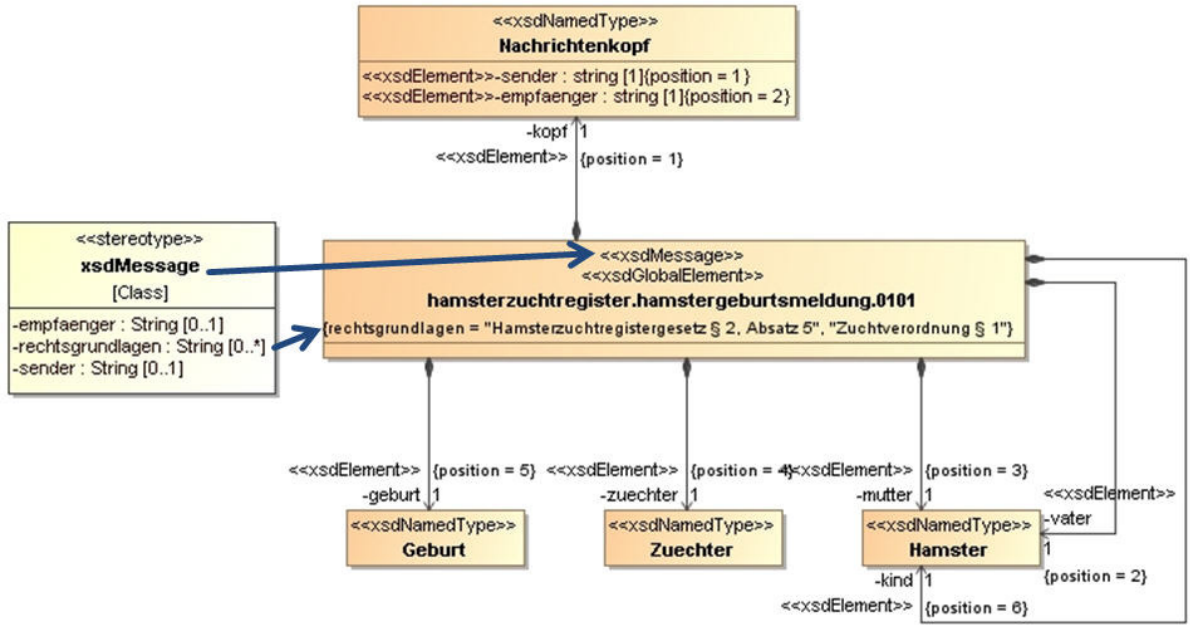
```
<xs:complexType name="Beispiel">
  <xs:sequence>
    <xs:element name="ergebnis">
      <xs:complexType>
        <xs:choice>
          <xs:element name="grundFuerAusfallDesWettbewerbs" type="xs:string"/>
          <xs:element name="teilnehmer" type="xhz:Teilnehmer" maxOccurs="unbounded"/>
        </xs:choice>
      </xs:complexType>
    </xs:element>
    ...
  </xs:sequence>
</xs:complexType>
```

Verwendung mit "xsdLocalStructure":

```
<xs:complexType name="Beispiel">
  <xs:sequence>
    <xs:choice>
      <xs:element name="grundFuerAusfallDesWettbewerbs" type="xs:string"/>
      <xs:element name="teilnehmer" type="xhz:Teilnehmer" maxOccurs="unbounded"/>
    </xs:choice>
    ...
  </xs:sequence>
</xs:complexType>
```

Zu dem Stereotypen sind keine Eigenschaften definiert.

4.2.18 xsdMessage



Anwendbar für: Class (UML-Klassen)

Der Stereotyp "xsdMessage" kennzeichnet die Instanzen dieser UML-Klasse als Nachrichten im XÖV-Sinne, also einer elektronischen Datenübermittlung innerhalb und mit der Verwaltung.

Da XÖV-Nachrichten XML-Instanzen darstellen, sind diese mittels des Stereotyps "xsdGlobalElement" immer als globale XML-Elemente zu definieren.

Eigenschaften von xsdMessage		
Eigenschaft	**Typ**	**Häufigkeit**
empfaenger	String	0 .. 1
rechtsgrundlagen	String	0 .. n
sender	String	0 .. 1

4.2.18.1 empfaenger (String)

Der Empfänger der Nachricht, z.B. "Zuchtverein"

Die Information dient nur zu Dokumentationszwecken und hat keine Auswirkung auf die Struktur des XML-Schemas: Es erfolgt eine Ausgabe in der Dokumentation des jeweiligen xs:element[1]-Eintrags im XML-Schema.

4.2.18.2 rechtsgrundlagen (String)

Die Rechtsgrundlage(n) wie beispielsweise Gesetze, Verordnungen oder Satzungen zu dieser Nachricht.

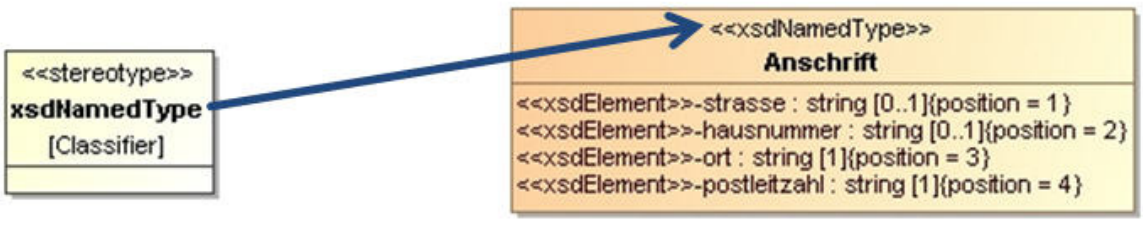
Die Information dient nur zu Dokumentationszwecken und hat keine Auswirkung auf die Struktur des XML-Schemas: Es erfolgt eine Ausgabe in der Dokumentation des jeweiligen xs:element[2]-Eintrags im XML-Schema.

4.2.18.3 sender (String)

Der Absender der Nachricht z.B. "Hamstermelderegister".

Die Information dient nur zu Dokumentationszwecken und hat keine Auswirkung auf die Struktur des XML-Schema: Es erfolgt eine Ausgabe in der Dokumentation des jeweiligen xs:element[3]-Eintrags im XML-Schema.

4.2.19 xsdNamedType



1.siehe http://www.w3.org/TR/xmlschema-1/#cElement_Declarations

2.siehe http://www.w3.org/TR/xmlschema-1/#cElement_Declarations

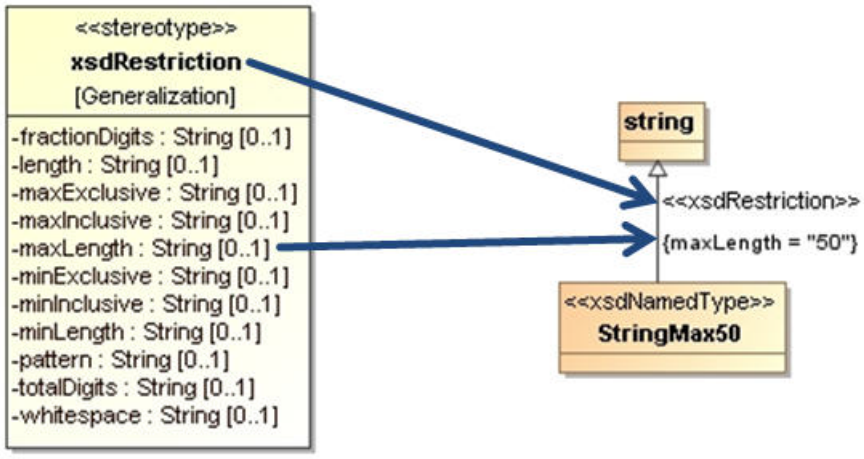
3.siehe http://www.w3.org/TR/xmlschema-1/#cElement_Declarations

Anwendbar für: Classifier (UML-Klassen, -Datentypen und -Aufzählungen)

Eine UML-Klasse, die mit "xsdNamedType" versehen ist, wird im XML-Schema als benannter Typ (xs:complexType[1] oder xs:simpleType[2]) umgesetzt.

Zu dem Stereotypen sind keine Eigenschaften definiert.

4.2.20 xsdRestriction



Anwendbar für: Generalization (UML-Vererbungsbeziehungen)

Wird eine UML-Generalisierung mit diesem Stereotyp gekennzeichnet, wird ein Hinzufügen von Informationen in der Unterklasse verboten. Die Instanzen der Unterklasse dürfen nur noch die Eigenschaften der Oberklasse verwenden, die explizit noch einmal in der Unterklasse aufgezählt werden. Die Vererbung wird hiermit eingeschränkt. Die Multiplizitäten (d. h. Häufigkeiten) der wiederverwendeten Eigenschaften dürfen zwar verändert, aber lediglich verschärft werden (z.B. von 0..1 zu 1 oder von 0..* zu 1..*).

Für Typen mit einfachem Inhalt können die als Eigenschaften abgebildeten Facetten[3] zur Einschränkung genutzt werden. Sie werden direkt im XML-Schema abgebildet.

Eigenschaften von xsdRestriction		
Eigenschaft	**Typ**	**Häufigkeit**
fractionDigits	String	0 .. 1
length	String	0 .. 1
maxExclusive	String	0 .. 1
maxInclusive	String	0 .. 1
maxLength	String	0 .. 1
minExclusive	String	0 .. 1

1.siehe http://www.w3.org/TR/xmlschema-1/#Complex_Type_Definitions

2.siehe http://www.w3.org/TR/xmlschema-1/#Simple_Type_Definitions

3.siehe http://www.w3.org/TR/xmlschema-2/#rf-facets

Eigenschaften von xsdRestriction		
Eigenschaft	**Typ**	**Häufigkeit**
minInclusive	String	0 .. 1
minLength	String	0 .. 1
pattern	String	0 .. 1
totalDigits	String	0 .. 1
whitespace	String	0 .. 1

4.2.20.1 fractionDigits (String)

xs:fractionDigits[1] definiert die maximal mögliche Anzahl von Ziffern nach dem Dezimalpunkt.

Zulässig ist die Angabe einer solchen Facette im XML-Schema nur für xs:decimal[2].

4.2.20.2 length (String)

xs:length[3] definiert eine feste Länge von Zeichen (allgemein) oder Bytes (xs:base64Binary bzw. xs:hexBinary).

Zulässig ist die Angabe einer solchen Facette im XML-Schema für die folgenden W3C-Datentypen[4]: xs:anyURI, xs:base64Binary, xs:ENTITY, xs:hexBinary, xs:ID, xs:IDREF, xs:language, xs:Name, xs:NCName, xs:NMTOKEN, xs:normalizedString, xs:NOTATION, xs:QName, xs:string, xs:token.

4.2.20.3 maxExclusive (String)

xs:maxExclusive[5] definiert einen nicht zu erreichenden Maximalwert.

Zulässig ist die Angabe einer solchen Facette im XML-Schema für die folgenden W3C-Datentypen[6]: xs:byte, xs:date, xs:dateTime, xs:decimal, xs:double, xs:duration, xs:float, xs:gDay, xs:gMonth, xs:gYear, xs:gYearMonth, xs:int, xs:integer, xs:long, xs:negativeInteger, xs:nonNegativeInteger, xs:nonPositiveInteger, xs:positiveInteger, xs:short, xs:time, xs:unsignedByte, xs:unsignedInt, xs:unsignedLong, xs:unsignedShort.

4.2.20.4 maxInclusive (String)

xs:maxInclusive[7] definiert einen erreichbaren Maximalwert.

1.siehe http://www.w3.org/TR/xmlschema-2/#rf-fractionDigits

2.siehe http://www.w3.org/TR/xmlschema-2/#decimal

3.siehe http://www.w3.org/TR/xmlschema-2/#rf-length

4.siehe http://www.w3.org/TR/xmlschema-2/#built-in-primitive-datatypes und http://www.w3.org/TR/xmlschema-2/#built-in-derived

5.siehe http://www.w3.org/TR/xmlschema-2/#rf-maxExclusive

6.siehe http://www.w3.org/TR/xmlschema-2/#built-in-primitive-datatypes und http://www.w3.org/TR/xmlschema-2/#built-in-derived

7.siehe http://www.w3.org/TR/xmlschema-2/#rf-maxInclusive

Zulässig ist die Angabe einer solchen Facette im XML-Schema für die folgenden W3C-Datentypen[1]: xs:byte, xs:date, xs:dateTime, xs:decimal, xs:double, xs:duration, xs:float, xs:gDay, xs:gMonth, xs:gYear, xs:gYearMonth, xs:int, xs:integer, xs:long, xs:negativeInteger, xs:nonNegativeInteger, xs:non-PositiveInteger, xs:positiveInteger, xs:short, xs:time, xs:unsignedByte, xs:unsignedInt, xs:un-signedLong, xs:unsignedShort.

4.2.20.5 maxLength (String)

xs:maxLength[2] definiert die maximale Länge von Zeichen (allgemein) oder Bytes (xs:base64Binary bzw. xs:hexBinary).

Zulässig ist die Angabe einer solchen Facette im XML-Schema für die folgenden W3C-Datentypen: xs:anyURI, xs:base64Binary, xs:ENTITY, xs:hexBinary, xs:ID, xs:IDREF, xs:language, xs:Name, xs:NCName, xs:NMTOKEN, xs:normalizedString, xs:NOTATION, xs:QName, xs:string, xs:token.

4.2.20.6 minExclusive (String)

xs:minExclusive[3] definiert einen nicht zu erreichenden Minimalwert.

Zulässig ist die Angabe einer solchen Facette im XML-Schema für die folgenden W3C-Datentypen: xs:byte, xs:date, xs:dateTime, xs:decimal, xs:double, xs:duration, xs:float, xs:gDay, xs:gMonth, xs:gYear, xs:gYearMonth, xs:int, xs:integer, xs:long, xs:negativeInteger, xs:nonNegativeInteger, xs:non-PositiveInteger, xs:positiveInteger, xs:short, xs:time, xs:unsignedByte, xs:unsignedInt, xs:un-signedLong, xs:unsignedShort.

4.2.20.7 minInclusive (String)

xs:minInclusive[4] definiert einen erreichbaren Minimalwert.

Zulässig ist die Angabe einer solchen Facette im XML-Schema für die folgenden W3C-Datentypen[5]: xs:byte, xs:date, xs:dateTime, xs:decimal, xs:double, xs:duration, xs:float, xs:gDay, xs:gMonth, xs:gYear, xs:gYearMonth, xs:int, xs:integer, xs:long, xs:negativeInteger, xs:nonNegativeInteger, xs:non-PositiveInteger, xs:positiveInteger, xs:short, xs:time, xs:unsignedByte, xs:unsignedInt, xs:un-signedLong, xs:unsignedShort.

4.2.20.8 minLength (String)

xs:minLength[6] definiert eine Mindestlänge von Zeichen (allgemein) oder Bytes (xs:base64Binary bzw. xs:hexBinary).

Zulässig ist die Angabe einer solchen Facette im XML-Schema für die folgenden W3C-Datentypen: xs:anyURI, xs:base64Binary, xs:ENTITY, xs:hexBinary, xs:ID, xs:IDREF, xs:language, xs:Name, xs:NCName, xs:NMTOKEN, xs:normalizedString, xs:NOTATION, xs:QName, xs:string, xs:token.

1.siehe http://www.w3.org/TR/xmlschema-2/#built-in-primitive-datatypes und http://www.w3.org/TR/xmlschema-2/#built-in-derived

2.siehe http://www.w3.org/TR/xmlschema-2/#rf-maxLength

3.siehe http://www.w3.org/TR/xmlschema-2/#rf-minExclusive

4.siehe http://www.w3.org/TR/xmlschema-2/#rf-minInclusive

5.siehe http://www.w3.org/TR/xmlschema-2/#built-in-primitive-datatypes und http://www.w3.org/TR/xmlschema-2/#built-in-derived

6.siehe http://www.w3.org/TR/xmlschema-2/#rf-minLength

4.2.20.9 pattern (String)

xs:pattern[1] definiert ein Muster, das auf einen String angewendet wird.

Zulässig ist die Angabe einer solchen Facette im XML-Schema für die folgenden W3C-Datentypen: xs:anyURI, xs:base64Binary, xs:boolean, xs:byte, xs:date, xs:dateTime, xs:decimal, xs:double, xs:duration, xs:ENTITY, xs:float, xs:gDay, xs:gMonth, xs:gYear, xs:gYearMonth, xs:hexBinary, xs:ID, xs:IDREF, xs:int, xs:integer, xs:language, xs:long, xs:Name, xs:NCName, xs:negativeInteger, xs:NMTOKEN, xs:nonNegativeInteger, xs:nonPositiveInteger, xs:normalizedString, xs:NOTATION, xs:positiveInteger, xs:QName, xs:short, xs:string, xs:time, xs:token, xs:unsignedByte, xs:unsignedInt, xs:unsignedLong, xs:unsignedShort.

4.2.20.10 totalDigits (String)

xs:totalDigits[2] definiert die maximale Anzahl von Dezimalziffern.

Zulässig ist die Angabe einer solchen Facette im XML-Schema für die folgenden W3C-Datentypen: xs:byte, xs:decimal, xs:int, xs:integer, xs:long, xs:negativeInteger, xs:nonNegativeInteger, xs:nonPositiveInteger, xs:positiveInteger, xs:short, xs:unsignedByte, xs:unsignedInt, xs:unsignedLong, xs:unsignedShort.

4.2.20.11 whitespace (String)

xs:whitespace[3] legt fest, wie Whitespace (z.B. Leerzeichen, Tabs, Linefeeds, Carriage Returns) behandelt werden sollen.

Mögliche Werte sind:

- preserve - Whitespace wird unverändert beibehalten
- replace - alle Whitespace-Zeichen werden durch ein Leerzeichen ersetzt
- collapse - am Anfang und am Ende stehende Whitespaces werden entfernt sowie zusammenhängende Whitespaces durch ein Leerzeichen ersetzt

Zulässig ist die Angabe einer solchen Facette im XML-Schema für die folgenden W3C-Datentypen: xs:ENTITY, xs:ID, xs:IDREF, xs:language, xs:Name, xs:NCName, xs:NMTOKEN, xs:normalizedString, xs:string, xs:token.

4.2.21 xsdSchema

Anwendbar für: Package (UML-Pakete)

1.siehe http://www.w3.org/TR/xmlschema-2/#rf-pattern

2.siehe http://www.w3.org/TR/xmlschema-2/#rf-totalDigits

3.siehe http://www.w3.org/TR/xmlschema-2/#rf-whiteSpace

Das so annotierte UML-Paket wird in ein XML-Schema übersetzt. Es besteht eine direkte Beziehung zwischen einer erzeugten XML-Schema-Datei und einem "xsdSchema"-Paket.

Die Umsetzung erfolgt als xs:schema[1] in einer XML-Schema-Datei.

Eigenschaften von xsdSchema		
Eigenschaft	**Typ**	**Häufigkeit**
namespace	String	0 .. 1
prefix	String	0 .. 1
schemaFile	String	1
schemaLocation	String	0 .. 1
version	String	0 .. 1

4.2.21.1 namespace (String)

Der Namensraum für dieses Schema.

Bei der Erzeugung des XML-Schemas wird dieser Wert für das "targetNamespace"-Attribut des xs:schema-Elements[2] verwendet.

4.2.21.2 prefix (String)

Bei der Angabe der Eigenschaft wird dessen Wert für den XML-Namensraum dieses xs:schema-Elements[3] als XML-Präfix verwendet.

4.2.21.3 schemaFile (String)

Der physische Dateiname des XML-Schemas ohne eine Pfadangabe, der zur Benennung des XML-Schemas auf Dateiebene herangezogen wird.

4.2.21.4 schemaLocation (String)

Der physische Speicherort (URL inklusive Dateiname), an dem das Schema später gefunden werden kann. Dieses Schema-Attribut wird auf xs:import[4] und xs:include[5] abgebildet.

4.2.21.5 version (String)

Die Version des XML-Schemas.

Dieser Wert wird auf das Attribut "version" des xs:schema-Elements[6] abgebildet.

1.siehe http://www.w3.org/TR/xmlschema-1/#Schemas

2.siehe http://www.w3.org/TR/xmlschema-1/#Schemas

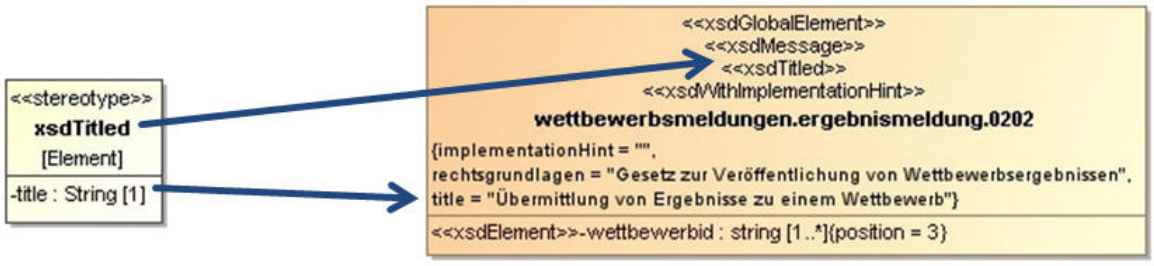
3.siehe http://www.w3.org/TR/xmlschema-1/#Schemas

4.siehe http://www.w3.org/TR/xmlschema-1/#composition-schemaImport

5.siehe http://www.w3.org/TR/xmlschema-1/#compound-schema

6.siehe http://www.w3.org/TR/xmlschema-1/#Schemas

4.2.22 xsdTitled



Anwendbar für: Element (alle UML-Elemente)

Ein mit diesem Stereotyp gekennzeichnetes UML-Element (z.B. eine UML-Klasse) erhält in der Dokumentation des XÖV-Standards einen anderen Titel als seinen Namen im UML-Modell (z.B. den UML-Klassennamen). Dies ist insbesondere dann sinnvoll, wenn der Titel in der Dokumentation beispielsweise Leerzeichen oder Umlaute enthalten soll.

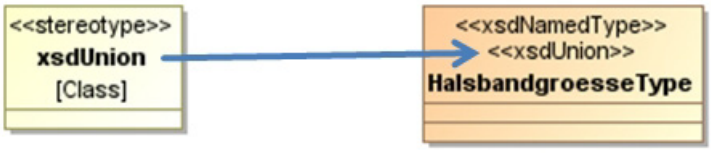
Eigenschaften von xsdTitled		
Eigenschaft	**Typ**	**Häufigkeit**
title	String	1

4.2.22.1 title (String)

Der Prosa-Titel in der Dokumentation.

Im XML-Schema wird die Eigenschaft in xs:annotation des jeweiligen XML-Schema-Bestandteils abgebildet.

4.2.23 xsdUnion



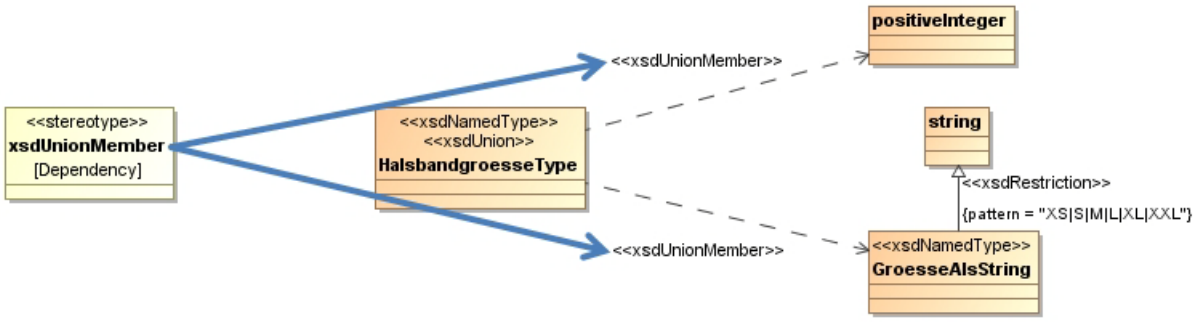
Anwendbar für: Class (UML-Klassen)

Angewendet auf eine UML-Klasse bewirkt dieser Stereotyp, dass eine xs:union[1] als Inhaltsmodell des simple Types festgelegt wird. Die Mitglieder der Union werden über Abhängigkeiten, die mit dem Stereotyp "xsdUnionMember" annotiert sind, modelliert.

Zu dem Stereotypen sind keine Eigenschaften definiert.

1.siehe http://www.w3.org/TR/xmlschema-1/#Simple_Type_Definitions

4.2.24 xsdUnionMember

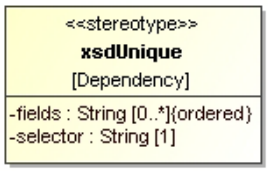


Anwendbar für: Dependency (UML-Abhängigkeitsbeziehung)

Mit diesem Stereotypen werden die Mitglieder einer xs:union festgelegt. Der simple Type an der Pfeilspitze der Abhängigkeit wird als möglicher Datentyp in die Union aufgenommen.

Zu dem Stereotypen sind keine Eigenschaften definiert.

4.2.25 xsdUnique



Anwendbar für: Dependency (UML-Abhängigkeitsbeziehungen)

Eine mit diesem Stereotyp gekennzeichnete UML-Abhängigkeit modelliert ein xs:unique-Constraint[1] in XML. Der Name der UML-Abhängigkeit wird als Name des xs:unique[2] verwendet.

Die Anwendung auf eine UML-Abhängigkeit im UML-Modell erfolgt analog zum Stereotyp "xsdKey".

Eigenschaften von xsdUnique		
Eigenschaft	**Typ**	**Häufigkeit**
fields	String	0 .. n
selector	String	1

4.2.25.1 fields (String)

Die Pfadausdrücke für die xs:field-Elemente[3].

1.siehe http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions

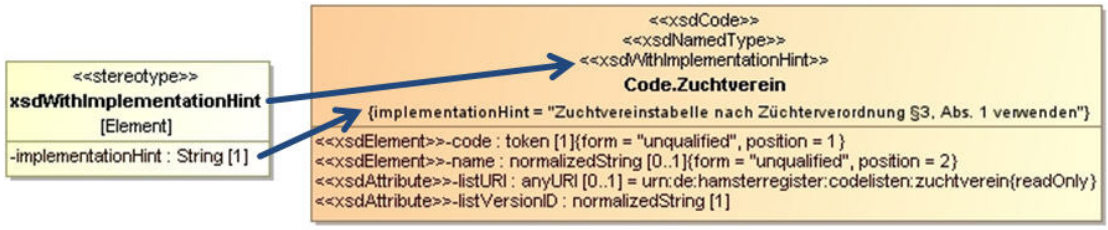
2.siehe http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions

3.siehe http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions

4.2.25.2 selector (String)

Der Pfadausdruck für das xs:selector-Element[1].

4.2.26 xsdWithImplementationHint



Anwendbar für: Element (alle UML-Elemente)

Ein mit diesem Stereotyp gekennzeichnetes UML-Element erhält in der Dokumentation neben der normalen Beschreibung einen Abschnitt mit Implementierungshinweisen.

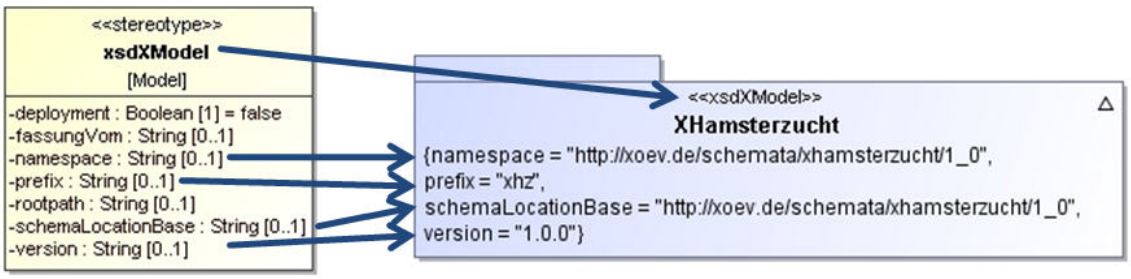
Eigenschaften von xsdWithImplementationHint		
Eigenschaft	**Typ**	**Häufigkeit**
implementationHint	String	1

4.2.26.1 implementationHint (String)

Der Implementierungshinweis zu dem XML-Bestandteil.

Im XML-Schema wird die Eigenschaft in xs:annotation des jeweiligen XML-Schema-Bestandteils abgebildet.

4.2.27 xsdXModel



Anwendbar für: Model (UML-Modelle)

Das mit diesem Stereotyp annotierte UML-Modell stellt den XÖV-Standard dar und enthält alle "xsd-Schema"-Pakete des XÖV-Standards sowie weitere relevante Informationen (z.B. xsdImport, xsdInclude).

1.siehe http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions

Eigenschaften von xsdXModel		
Eigenschaft	**Typ**	**Häufigkeit**
deployment	Boolean	1
fassungVom	String	0 .. 1
namespace	String	0 .. 1
prefix	String	0 .. 1
rootpath	String	0 .. 1
schemaLocationBase	String	0 .. 1
version	String	0 .. 1

4.2.27.1 deployment (Boolean)

Der Schalter zur Angabe, ob der Standard sich noch in der lokalen Entwicklung befindet (false) oder das Deployment mit der Veröffentlichung des Standards (true) bevorsteht und somit auch die XML-Schemata über öffentlich zugängliche URLs angesprochen werden können.

Bei Angabe des Wertes "true" wird im Normalfall für das Attribut "schemaLocation" für xs:import[1] und xs:include[2] der Wert der Eigenschaft "schemaLocation" im Stereotyp "xsdSchema" verwendet. Ist "schemaLocation" für das "xsdSchema"-Paket nicht angegeben so wird eine Kombination der Eigenschaftswerte aus "schemaFile" ("xsdSchema"-Paket) und "schemaLocationBase" ("xsdXModel"-Modell) gebildet.

Bei Angabe des Wertes "false" wird für das Attribut "schemaLocation" für xs:import[3] und xs:include[4] der Wert der Eigenschaft "schemaFile" im Stereotyp "xsdSchema" übernommen.

4.2.27.2 fassungVom (String)

Datum der Fassung.

Die Information dient nur zu Dokumentationszwecken und hat keine Auswirkung auf die Struktur des XML-Schema.

4.2.27.3 namespace (String)

Der gültige Namensraum für alle "xsdSchema"-Pakete des Standards, sofern für das einzelne "xsdSchema"-Paket die gleichnamige Stereotyp-Eigenschaft nicht mit einem eigenen Wert versehen ist.

Bei der Erzeugung der XML-Schemata wird dieser Wert für das "targetNamespace"-Attribut des xs:schema-Elements[5] verwendet.

1.siehe http://www.w3.org/TR/xmlschema-1/#composition-schemaImport

2.siehe http://www.w3.org/TR/xmlschema-1/#compound-schema

3.siehe http://www.w3.org/TR/xmlschema-1/#composition-schemaImport

4.siehe http://www.w3.org/TR/xmlschema-1/#compound-schema

5.siehe http://www.w3.org/TR/xmlschema-1/#Schemas

4.2.27.4 prefix (String)

Der gültige Prefix für alle "xsdSchema"-Pakete des Standards, sofern für das einzelne "xsdSchema"-Paket die gleichnamige Stereotyp-Eigenschaft nicht mit einem eigenem Wert versehen ist.

Bei der Angabe der Eigenschaft wird dessen Wert für den Namespace als XML-Präfix für die xs:schema-Elemente[1] verwendet.

4.2.27.5 rootpath (String)

Der absolute Pfad zum jeweiligen Projektverzeichnis des XÖV-Standards für den XGenerator. Der Parameter bestimmt somit, an welchen Ort die generierten Dateien geschrieben werden.

4.2.27.6 schemaLocationBase (String)

Der Basispfad der SchemaLocations (z.B. "http://xoev.de/schemata/xhamsterzucht/1_0").

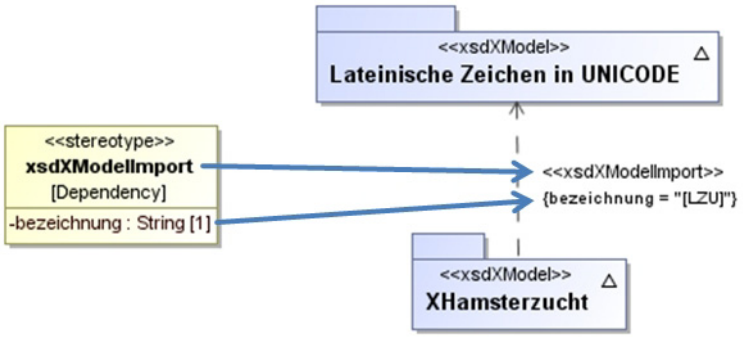
Der physische Speicherort (URL exklusive Dateiname), an dem die Schemata später gefunden werden können. Bildet zusammen mit der Eigenschaft "schemaFile" (z.B. "xhz-baukasten.xsd") des Stereotyps "xsdSchema" auf das Schema-Attribut "schemaLocation" (z.B. "http://xoev.de/schemata/xhamster-zucht/1_0 xhz-baukasten.xsd") für xs:import[2] und xs:include[3] ab - sofern für das einzelne Schema kein konkreter Wert für die Eigenschaft "schemaLocation" im Stereotyp "xsdSchema" angegeben ist.

4.2.27.7 version (String)

Die gültige Version für alle XML-Schemata des Standards, sofern für das einzelne "xsdSchema"-Paket die gleichnamige Stereotyp-Eigenschaft nicht mit einem eigenem Wert versehen ist.

Der Wert bildet auf das "version"-Attribut der zugeordneten xs:schema-Elemente[4] ab.

4.2.28 xsdXModelImport



Anwendbar für: Dependency (UML-Abhängigkeiten)

Dieser Stereotyp wird für die Dokumentation von Datentypen aus eingebundenen externen Modellen benötigt. Bei der Verwendung eines externen Datentyps kann die Dokumentation auf die hier definierten Daten zurückgreifen und so einen Verweis auf ein externes Dokument erzeugen.

1.siehe http://www.w3.org/TR/xmlschema-1/#Schemas

2.siehe http://www.w3.org/TR/xmlschema-1/#composition-schemaImport

3.siehe http://www.w3.org/TR/xmlschema-1/#compound-schema

4.siehe http://www.w3.org/TR/xmlschema-1/#Schemas

Der Stereotyp hat keine Auswirkungen auf die generierten XML-Schemata.

Eigenschaften von xsdXModelImport		
Eigenschaft	**Typ**	**Häufigkeit**
bezeichnung	String	1

4.2.28.1 bezeichnung (String)

Bei Verweisen verwendeter Bezeichner. Beispiel: [XOEV-Basisdatentypen]

4.3 XÖV-Basisdatentypen

Dieser Abschnitt beschreibt die XÖV-Basisdatentypen als Bestandteil des XÖV-UML-Profils. Die Datentypen haben die Aufgabe, UML-Elemente (z.B. UML-Klassenattribute) in ihrem Wertebereich einzuschränken. Als XÖV-Basisdatentypen werden einerseits W3C-Datentypen eingesetzt, andererseits weitere spezifische Datentypen, welche auf W3C-Datentypen aufbauen. Die spezifischen Datentypen umfassen in der aktuellen Fassung lediglich den Datentyp "Code" für die Übermittlung von Daten, die auf Codelisten basieren (siehe auch Abschnitt 6 auf Seite 79), der vormals enthaltene Datentyp String.Latin wurde in einen eigenständigen Standard überführt. Weitere Datentypen – z.B. für die Übermittlung von Zeitangaben – werden in Folgeversionen des XÖV-Handbuchs eingeführt.

Die XML-Schema-Repräsentation der spezifischen Basisdatentypen sind unter http://xoev.de/schemata/basisdatentypen/1_1/xoev-basisdatentypen.xsd mit dem Namensraum `http://xoev.de/schemata/basisdatentypen/1_1` verfügbar.

4.3.1 W3C-Datentypen

Die W3C-XML-Schema-Spezifikation[1] definiert unterschiedliche Datentypen. Die in der nachfolgenden Tabelle aufgeführten Datentypen werden im Rahmen von XÖV zu den XÖV-Basisdatentypen gezählt.

Hinweis: Die lexikalischen Festlegungen zu Zeitangaben[2] ermöglichen die Angabe einer Zeitzone. Diese Angabe ist aber nicht mandantorisch. Wird die Zeitzone nicht angegeben, so kann ein Ereignis um 09:07 Uhr zeitlich vor einem Ereignis um 08:41 Uhr stattgefunden haben, falls die Ereignisse in Ländern mit unterschiedlichen Zeitzonen stattfanden. Es wird daher empfohlen, Zeitangaben für eine spätere Vergleichbarkeit mit anderen Zeitangaben immer mit einer Zeitzone zu versehen.

---

anyURI	language
base64Binary	long
boolean	Name
byte	NCName

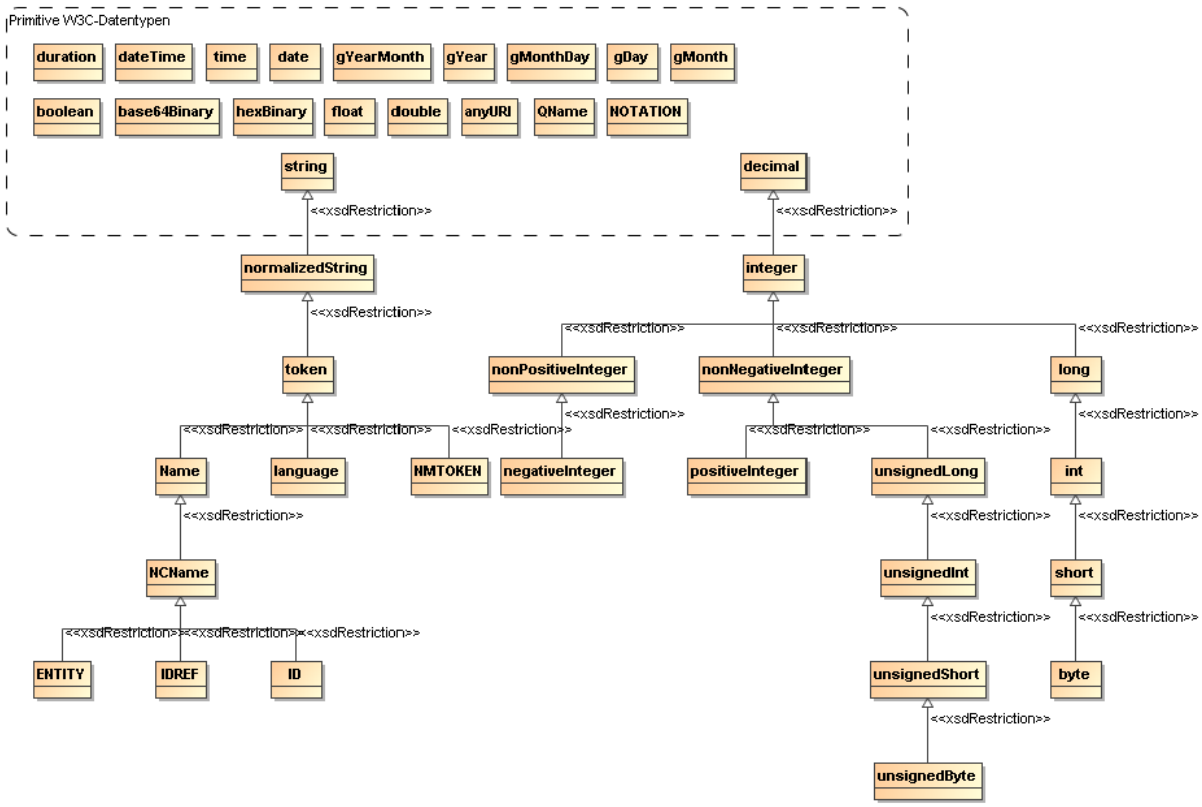
1.siehe http://www.w3.org/TR/xmlschema-2/

2.Zeitangaben sind date, dateTime, duration, gDay, gMonth, gMonthDay, gYear, gYearMonth und time.

date	negativeInteger
dateTime	NMTOKEN
decimal	nonNegativeInteger
double	nonPositiveInteger
duration	normalizedString
ENTITY	NOTATION
float	positiveInteger
gDay	QName
gMonth	short
gMonthDay	string
gYear	time
gYearMonth	token
hexBinary	unsignedByte
ID	unsignedInt
IDREF	unsignedLong
int	unsignedShort
integer	

Die W3C-Datentypen lassen sich im Wesentlichen in drei Gruppen einteilen: primitive Datentypen, vom primitiven Datentyp "string" abgeleitete Datentypen und vom primitiven Datentyp "decimal" abgeleitete Datentypen (siehe nachfolgende Abbildung).

Bild 4-1 W3C-Datentypen



4.3.2 Datentyp Code

Der folgende Abschnitt betrachtet den Ursprung des XÖV-Basisdatentyps Code, seinen Aufbau in UML sowie die Darstellung im XML-Schema.

4.3.2.1 Orientierung am UN/CEFACT-Datentyp Code

Der XÖV-Basisdatentyp Code ist nach dem Vorbild des entsprechenden Datentyps der UN/CEFACT gebildet, weicht aber in einigen Punkten von diesem ab.

Zum Thema Code hat die UN/CEFACT die folgende Definition herausgegeben:

> *[A code is a] character string (letters, figures or symbols) that for brevity and/or language independence may be used to represent or replace a definitive value or text of an attribute.*[1]

Codes werden im Kontext von Codelisten definiert. In einer Codeliste wird für einen Begriff (z. B. "Familienstand") ein vordefinierter Umfang von Werten festgelegt (z. B. "ledig", "verheiratet", ...), auf den sich die Kommunikationspartner durch den Einsatz der Codeliste im Nachrichtenaustausch beziehen. Dabei wird jeder der vordefinierten Werte durch einen Code (z.B. "ld", "vh") identifiziert.

1."Core Component Data Type Catalogue", Version 3.0, 29. September 2009, S.31 (http://www.uncefactforum.org/ATG/Documents/ATG/Downloads/CCTS%202p01%20Data%20Type%20Catalogue%20Version%202p1.pdf)

Bei der Entwicklung des UN/CEFACT-Datentyps Code wurde davon ausgegangen, dass Codes grundsätzlich vollständig dokumentiert und eigenständig (self-contained) übermittelt werden sollen; alle Metainformationen eines übermittelten Codes, z. B. bezüglich seiner Bedeutung, die zu ihm gehörige Codeliste, Herausgeber der Codeliste, Bereitstellungsort der Codeliste, werden also stets im Zusammenhang des Codes übertragen.

Beispiel: Code "000", das Land Deutschland bezeichnend, definiert in der Codeliste "Staatsangehörigkeits- und Gebietsschlüssel", herausgegeben vom Statistischen Bundesamt, in der Fassung vom 12.12.2008, verfügbar auf einer bestimmten Webseite usw.

Die UN/CEFACT bildet diese Informationen zu einem Code über die folgenden Eigenschaften ab:

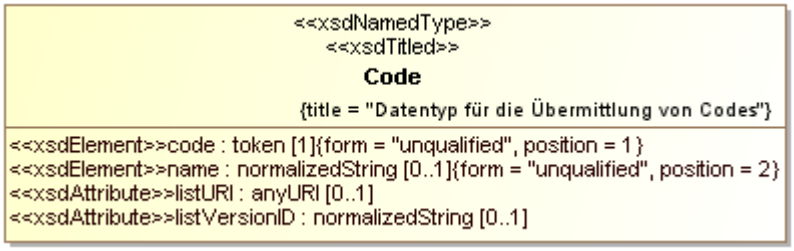
- Code List Agency Identifier (listAgencyID)
- Code List Agency Name (listAgencyName)
- Code List Name (listName)
- Code List Identifier (listID)
- Code List Scheme Uniform Resource Identifier (listSchemaURI)
- Code List Uniform Resource Identifier (listURI)
- Code List Version Identifier (listVersionID)

Darüber hinaus können neben dem eigentlichen Code auch sein ausgeschriebener Wert und die für die Formulierung verwendete Sprache in eine Nachricht eingetragen werden:

- Code Name (name)
- Code Language Identifier (languageID)

4.3.2.2 UML-Repräsentation von Code

Der Basisdatentyp Code orientiert sich an dem beschriebenen UN/CEFACT-Datentyp, schränkt aber die Auswahl der relevanten Metainformationen ein. Im Unterschied zum UN/CEFACT-Datentyp macht der Entwurf des XÖV-Datentyps Code bestimmte Annahmen über den Kontext der Verwendung eines Codes: Er geht davon aus, dass alle in XÖV-Standards zu verwendenden Codelisten eindeutig durch ein Paar aus Uniform Resource Identifier (URI) und Versionsangabe identifiziert werden können und dass die Metainformationen zur Codeliste auf der Basis dieses Paars elektronisch aus einem bereitgestellten Verzeichnis (dem XRepository) abgerufen werden können. Der XÖV-Entwurf geht weiter davon aus, dass die Liste der Codes, mit ihren Werten und weiteren relevanten Informationen, in der Regel ebenfalls an dieser Stelle abrufbar ist. So kann der Empfänger die bezeichnete Liste in der richtigen Version auffinden und den Code interpretieren. Darüber hinaus sieht der XÖV-Datentyp Code die Möglichkeit vor, optional den Wert des übermittelten Codes zu ergänzen (z. B. für den Schlüssel "000" den Wert "Deutschland"). Wie Metainformationen und Codelisten im XRepository bereitgestellt werden, wird in Abschnitt 6.1 auf Seite 79 näher erläutert.



Die folgende Tabelle zeigt die Typ-Eigenschaften, die in der Datenübermittlung eines Codes im Kontext XÖV genutzt werden müssen bzw. können:

Eigenschaft	Datentyp	Multiplizität	Beschreibung
code	xs:token	1	Der Schlüssel aus der Codeliste.
name	xs:normalizedString	0..1	Der Wert des Schlüssels.
listURI	xs:anyURI	0..1	Der URI, welcher die Codeliste, auf deren Basis der übermittelte Schlüssel zu interpretieren ist, eindeutig identifiziert (die Version der bezeichneten Liste ist dadurch noch nicht festgelegt).
listVersionID	xs:normalizedString	0..1	Die Version der identifizierten Codeliste.

4.3.2.3 XML-Schema-Repräsentation von Code

Der XÖV-Basisdatentyp Code wird im XML-Schema als *complexType* mit *complexContent* abgebildet. Ein Datentyp für eine konkrete Codeliste (z. B. für die Codeliste Familienstand oder die Codeliste Staat) ist auf UML-Ebene als Restriktion vom XÖV-Basisdatentyp Code abzuleiten und ist dann auf XML-Schema-Ebene ebenfalls als complexType dargestellt. Innerhalb des complexType werden der Code und sein Wert durch jeweils eigene XML-Elemente (code und name) und die Metadaten als XML-Attribute (listURI und listVersionID) repräsentiert.

Hier wird vom UN/CEFACT-Datentyp Code abgewichen, wo der Code als Wert des simpleContent und sein Wert als XML-Attribut dargestellt werden. Motivation dieser Entscheidung ist der Bedarf innerhalb von XÖV-Standards, Integritäts-Constraints (xs:key, xs:unique, xs:keyref) einzubinden, welche nur auf Codes als Elemente referenzieren können.

Auch ist innerhalb der UN/CEFACT-Konzeption für den Typ Code ein *complexType-Pendant* auf XML-Schema-Ebene nicht vorgesehen. Für den XÖV-Basisdatentyp Code wird ein entsprechender Bedarf aus softwaretechnischen Gründen gesehen. Die Umsetzung der Verarbeitung von Codelisten in z. B. aus XML-Schema generierten Java-Klassen wird erleichtert, wenn auch die Basisklasse als XML-Schema-Konstrukt verfügbar ist.

```
<xs:complexType name="Code">
  <xs:annotation>
    <xs:documentation>XÖV-Basisdatentyp Code</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="code" type="xs:token" form="unqualified"/>
    <xs:element name="name" type="xs:normalizedString" minOccurs="0" form="unqualified"/>
  </xs:sequence>
  <xs:attribute name="listURI" type="xs:anyURI" use="optional"/>
  <xs:attribute name="listVersionID" type="xs:normalizedString" use="optional"/>
</xs:complexType>
```

4.3.3 Datentyp String.Latin

Der XÖV-Datentyp String.Latin basiert auf dem internationalen Unicode-Standard, schränkt diesen aber auf die "Lateinischen Zeichen" ein. Damit wird folgende Vorgabe für die Registerführung und Datenverarbeitung in Deutschland umgesetzt: "Daten sind in lateinischer Schrift zu erfassen, diakritische Zeichen sind unverändert wiederzugeben. Dabei ist der Zeichensatz nach ISO-/IEC 10646 (der Unicode Standard) zu Grunde zu legen". Er wird im eigenständigen Standard *"Lateinische Zeichen in Unicode"* (verfügbar unter http://xoev.de/latinchars/) beschrieben. In diesem Standard wird auch der Datentyp in XML Schema und UML beschrieben.

5. XÖV-Namens- und Entwurfsregeln

Die Namens- und Entwurfsregeln (engl.: *"Naming and Design Rules"* - NDR) für XÖV-Standards legen die technische Ausgestaltung von XÖV-Standards fest. Ihre Verankerung in einem XÖV-Standard ist in K-10 auf Seite 18 beschrieben und bildet die Grundlage für die Abbildung eines UML-Modells auf ein mit dem XÖV-UML-Profil (siehe Abschnitt 4 auf Seite 25) profiliertem XÖV-UML-Modell.

Die Verbindlichkeitsstufen **Muss, Soll** und **Empfehlung** für die Namens- und Entwurfsregeln entsprechen den Festlegungen für die XÖV-Konformitätskriterien (Abschnitt 2 auf Seite 13).

Hinweis: Die Einhaltung der **Muss-** und **Soll-Regeln** ist relevant für die Einstufung eines Standards als XÖV-Standard (siehe Abschnitt 2 auf Seite 13). Die Beachtung dieser Regeln durch die Entwickler eines XÖV-Standards ist somit notwendig.

Das XÖV-Produktionszubehör stellt, wenn technisch möglich, die Einhaltung der **Muss**- und **Soll-Regeln** sowie der **Empfehlungen** sicher. Dies geschieht durch Automatismen in den XÖV-XSD-Vorlagen und Invariantenüberprüfungen (XÖV-Invarianten) bezüglich des UML-Modells. Der Umfang der automatisierten Sicherstellung durch das XÖV-Produktionszubehör wird im Folgenden für jede Regel einzeln ausgewiesen.

In den beiden folgenden Abschnitten werden die Regeln und Empfehlungen in Form

- einer thematisch kategorisierten Übersichtstabelle sowie
- detaillierten Beschreibungen mit Erläuterungen, Begründungen, Beispielen und Angaben zur technischen Prüfbarkeit

dargestellt.

5.1 Übersicht über die Regeln und Empfehlungen

Nr.	Verbindlichkeit	Kurzbeschreibung	Seite
Strukturen und Inhalte			
NDR-1	Muss	Konsistente Strukturen und Inhalte in UML-Modell und XML-Schemata	63
NDR-2	Muss	Hauptstruktur des UML-Modells	63
NDR-3	Muss	Nachrichten als globale Elemente	64
NDR-4	Soll	Erlaubte Einbindungsarten für Codelisten	64

Nr.	Verbindlichkeit	Kurzbeschreibung	Seite
NDR-5	Empfehlung	Detaillierte Struktur des UML-Modells	65
NDR-6	Empfehlung	Nutzung von XML-Attributen und XML-Elementen	66
NDR-7	Empfehlung	XML-Wildcard-Elemente mit Namensraum	67
NDR-8	Empfehlung	Eindeutige versionsübergreifende Codes in Codelisten	67
NDR-9	Empfehlung	Umgang mit unscharfen Codelisten-Einträgen und nicht abgeschlossenen Codelisten	67
Namen			
NDR-10	Muss	Konsistente Namen in UML-Modell und XML-Schemata	68
NDR-11	Soll	Erlaubte Zeichen für Namen	68
NDR-12	Soll	Erlaubte Zeichen für Klassifikationen in Namen	69
NDR-13	Soll	Eindeutige versionsübergreifende Namen von Nachrichten	69
NDR-14	Empfehlung	Namen in deutscher Sprache	69
NDR-15	Empfehlung	Groß- und Kleinschreibung von (und in zusammengesetzten) Namen	70
NDR-16	Empfehlung	Namensstruktur von globalen Elementen	70
NDR-17	Empfehlung	Eindeutige versionsübergreifende Nummern in Namen von Nachrichten	71
NDR-18	Empfehlung	Namen von XML-Schema-Dateien	72
Dokumentation			
NDR-19	Soll	Dokumentation in deutscher Sprache	72
NDR-20	Empfehlung	Dokumentation der Rechtsgrundlagen	72
NDR-21	Empfehlung	Codenamen für Codelisten-Einträge	73
Wiederverwendung			
NDR-22	Muss	Unveränderte Übernahme von XÖV-Codelisten	73
NDR-23	Muss	Umgang mit Restriktionen über unterschiedliche Namensräume	73
NDR-24	Soll	Wiederverwendung generischer Nachrichten-Eigenschaften	74
NDR-25	Empfehlung	Abgrenzung der Wiederverwendung durch Komposition und Restriktion	74
NDR-26	Empfehlung	Physische Speicherorte von XML-Schemata als URL	75
NDR-27	Empfehlung	Verwendung von Original-Namespace-Präfixen bei Schema-Importen	75
Technik und Infrastruktur			
NDR-28	Muss	Valide W3C-XML-Schemata	76
NDR-29	Muss	Identifizierende Namensräume	76
NDR-30	Muss	Versionierung der Schemata	76
NDR-31	Soll	Namensräume mit Versionen	77

5.2 Detaillierte Beschreibung der Regeln und Empfehlungen

Die im Folgenden aufgeführten Regeln sind analog zur Kategorisierung innerhalb der Übersichtstabelle geordnet.

5.2.1 Strukturen und Inhalte

Die Nutzung einer einheitlichen Struktur des Standards und die Aufteilung seiner Inhalte erlaubt die Verarbeitung des UML-Modells durch das XÖV-Produktionszubehör sowie einen schnellen Einstieg in einen anderen XÖV-Standard beispielsweise im Zuge der Prüfung einer möglichen Integration in den eigenen Standard.

NDR-1 (MUSS): Konsistente Strukturen und Inhalte in UML-Modell und XML-Schemata

Die Strukturen und Inhalte des UML-Modells eines XÖV-Standards müssen genau denen der erzeugten XML-Schemata entsprechen.

Erläuterung:

Die mit dem XGenerator aus dem XMI-Export des UML-Modells erzeugten XML-Schemata müssen zur Einhaltung der technischen XÖV-Konformität mit den durch die XÖV-Koordination bereitgestellten XÖV-XSD-Vorlagen generiert werden. Die XÖV-XSD-Vorlagen gewährleisten eine korrekte, konsistente und eindeutige Übersetzung des UML-Modells in XML-Schema. Die automatisiert erstellten XML-Schemata dürfen nachträglich nicht manuell verändert werden. Die Zuordnung der UML-Bestandteile zu den jeweiligen Entsprechungen im XML-Schema erfolgt unter Anwendung des XÖV-UML-Profils (siehe Abschnitt 4 auf Seite 25).

Begründung:

Eine transparente Übersetzung des UML-Modells in XML-Schema, ist die Grundlage für interoperable Standards. Sie gewährleistet eine nachvollziehbare Modellierung, da alle Inhalte und Strukturen eines XÖV-Standards ohne Ausnahme auf ihre Umsetzung in XML-Schema schließen lassen.

Prüfung:

Die Einhaltung dieser Regel wird durch das XÖV-Produktionszubehör sichergestellt: XÖV-XSD-Vorlagen.

NDR-2 (MUSS): Hauptstruktur des UML-Modells

Das UML-Modell eines XÖV-Standards muss auf oberster Ebene einer definierten Struktur folgen, welche externe Modelle von den Inhalten des eigenen Standards unterscheidet.
Die W3C-Datentypen, die Bestandteil der XÖV-Basisdatentypen sind, sind im UML-Modell in einem vorgegebenen UML-Paket abgebildet.

Erläuterung:

Das UML-Modell eines XÖV-Standards teilt sich auf der obersten Ebene mindestens in die folgenden beiden Bereiche:

a) UML-Modell mit dem Namen des XÖV-Standards und dem Stereotyp `xsdXModel`
b) UML-Paket mit dem Namen "Externe Modelle" und ohne einen im XÖV-Profil angegebenen Stereotypen

Das UML-Modell eines XÖV-Standards darf außerhalb des Pakets "Externe Modelle" und neben

dem Modell, das den eigenen XÖV-Standard selbst repräsentiert, keine weiteren Modelle mit dem Stereotypen `xsdXModel` enthalten. Das Modell aus a) enthält die `xsdSchema`-Pakete, aus denen die XML-Schemata des XÖV-Standards erzeugt werden (siehe auch NDR-5 auf Seite 65). Das Paket aus b) kann Modelle (d. h. mit dem Stereotyp `xsdXModel` gekennzeichnete UML-Modelle) externer Standards enthalten, aus denen Bestandteile in dem XÖV-Standard wiederverwendet werden, deren Inhalte jedoch nicht zum XÖV-Standard selbst gehören. Für externe Modelle werden keine XML-Schemata generiert, da sie bereits als XML-Schemata vorliegen. Über die Eigenschaft "schemaLocation" dieser externen XML-Schemata (Stereotyp `xsdSchema` oder `xsdXModel`) kann dann beispielsweise auf die entsprechenden Namensräume für den Import in die Standard-eigenen XML-Schemata zugegriffen werden. Die zu den XÖV-Basisdatentypen gehörenden W3C-Datentypen sind im UML-Modell als UML-Klassen innerhalb eines UML-Pakets mit dem Namen "W3C Data Types" abgebildet.

Begründung:

Die Abgrenzung des XÖV-Standards (d. h. seines UML-Modells) von ggf. weiteren eingebundenen Modellen ermöglicht die automatisierte Identifizierung der zu generierenden XML-Schemata innerhalb des Generierungsprozesses des XGenerators.

Beispiel:

Prüfung:

Die Einhaltung dieser Regel wird durch das XÖV-Produktionszubehör sichergestellt: XÖV-Invarianten.

NDR-3 (MUSS): Nachrichten als globale Elemente

Nachrichten eines XÖV-Standards müssen globale XML-Elemente sein.

Erläuterung:

Diese Regel bezieht sich auf XÖV-Nachrichten (Stereotyp `xsdMessage`). Diese müssen globale XML-Elemente darstellen (Stereotyp `xsdGlobalElement`).

Begründung:

Jede Nachricht muss eine XML-Instanz darstellen, die nur erzeugt werden kann, wenn die Nachricht als ein globales Element im XML-Schema definiert ist.

Prüfung:

Die Einhaltung dieser Regel wird durch das XÖV-Produktionszubehör sichergestellt: XÖV-Invarianten.

NDR-4 (SOLL): Erlaubte Einbindungsarten für Codelisten

Eine Codeliste soll ausschließlich als Standard, benannter, versionsfreier oder generischer Code-Typ in einen XÖV-Standard integriert werden.

Erläuterung:

Codelisten sollen wie in Abschnitt 6.2.3 auf Seite 91 beschrieben im Standard behandelt werden.

Begründung:

Die vier genannten Code-Typen leiten sich von dem XÖV-Basisdatentyp Code ab. Durch diese vier Varianten der Einbindung von Codelisten in einen XÖV-Standard, basierend auf dem einheitlichen XÖV-Basisdatentyp Code, ist die Nutzung einer einheitlichen Struktur und Benennung bezüglich der Übermittlung von Codes unter Wahrung der Flexibilität für verschiedene Anwendungskontexte gewährleistet.

Prüfung:

Die Einhaltung dieser Regel kann durch das XÖV-Produktionszubehör sichergestellt werden: XÖV-Invarianten.

NDR-5 (EMPFEHLUNG): Detaillierte Struktur des UML-Modells

Für die Gliederung des UML-Modells zum XÖV-Standard wird die Nutzung einer definierten Struktur empfohlen.

Erläuterung:

Das mit `xsdXModel` stereotypisierte UML-Modell des XÖV-Standards befindet sich – wie in NDR-2 auf Seite 63 beschrieben – auf der obersten Ebene des Gesamt-UML-Modells und beinhaltet die folgenden Elemente:

- *Basisdatentypen*: Paket, im Modell <*NameDesXÖVStandards*> (z. B. XHamsterzucht), mit dem Stereotypen `xsdSchema` versehen. Das entsprechende XML-Schema enthält alle technischen Typen, z. B. Einschränkungen von W3C-Datentypen.
- *Baukasten*: Paket, im Modell <*NameDesXÖVStandards*>, mit dem Stereotypen `xsdSchema` versehen. Das entsprechende XML-Schema enthält alle fachlichen Typen z. B. zur Wiederverwendung in den Nachrichten der unterschiedlichen Hauptgruppen.

- *Nachrichten* oder *Fachmodule*: Paket, im Modell <*NameDesXÖVStandards*>, ohne Stereotypen.
- (Mindestens eine) <*Hauptgruppe*>: Paket, im Paket *Nachrichten* oder *Fachmodule* mit dem Stereotypen `xsdSchema` versehen. Das entsprechende XML-Schema einer Hauptgruppe umfasst alle fachlich zusammengehörenden globalen Elemente (Stereotyp `xsdGlobalElement`) - insbesondere Nachrichten - sowie dazugehörige hauptgruppenspezifische Typen.

Neben den genannten Modellen und Paketen können weitere Bestandteile im UML-Modell abgebildet werden, z.B. zu den Verwaltungsprozessen, in deren Ablauf die globalen Elemente wie die Nachrichten (Stereotyp `xsdMessage`) zwischen den verschiedenen Kommunikationspartnern ausgetauscht werden. Weiterhin können unterhalb der aufgeführten Bestandteile weitere Strukturen in Form von Unterpaketen gebildet werden, um weitere Klassifikationen z.B. "Nachrichtenköpfe" unterhalb des "Baukasten"-Pakets vorzunehmen. Die technische Umsetzung in XML-Schemata auf der Grundlage der obligatorisch zu verwendenden XÖV-Vorlagen für XML-Schemata erfolgt nur für die Bestandteile des UML-Modells, für die ein XÖV-Stereotyp vergeben wurde.

Begründung:

Ziel ist die Etablierung einer einheitlichen Struktur der UML-Modelle und damit der erzeugten XML-Schemata für XÖV-Standards zur leichteren Vergleichbarkeit, Überprüfung und Einarbeitung mit anderen bzw. in andere Standards. Weiterhin ermöglicht die Einhaltung dieser Empfehlung die Nutzung der fakultativ zu verwendenden XÖV-Muster-Vorlagen für DocBook zur Erzeugung der Dokumentation – zu allen oben explizit aufgeführten Bestandteilen werden entsprechende DocBook-Fragmente generiert.

Beispiele für Inhalte:

`String.Max50` (Basisdatentypen), `Anschrift` (Baukasten), `hamsterzuchtregister.hamstergeburtsmeldung.0101` (Hauptgruppe "Hamsterzuchtregister")

Prüfung:

Die Einhaltung dieser Regel kann durch das XÖV-Produktionszubehör in der Form sichergestellt werden, dass der entsprechende Aufbau des UML-Modells vorhanden ist - hierbei wird jedoch nicht geprüft, ob die Inhalte den korrekten Strukturelementen zugeordnet sind: XÖV-Invarianten

NDR-6 (EMPFEHLUNG): Nutzung von XML-Attributen und XML-Elementen

a) Es wird empfohlen, fachliche Inhalte als XML-Elemente zu modellieren.

b) Es wird empfohlen, technische bzw. erläuternde Metadaten als XML-Attribute zu modellieren.

Erläuterung:

Während fachliche Inhalte in XML-Elementen (Stereotyp `xsdElement`) umgesetzt werden, können Metadaten, d.h. Informationen über die eigentlichen fachlichen Inhalte, durch Attribute (Stereotyp `xsdAttribute`) ausgedrückt werden.

Beispiel zu b):

Erstellungszeitpunkt einer Nachricht

NDR-7 (EMPFEHLUNG): XML-Wildcard-Elemente mit Namensraum

Für Wildcard-Elemente wird die Angabe eines Namensraums empfohlen.

Erläuterung:

Diese Regel bezieht sich auf XML-Schema-Elemente (xs:any), die beliebige Inhalte umfassen dürfen (Stereotyp `xsdAnyContents`). Für sie sollte ein Namensraum (Wert der Eigenschaft `namespace` des Stereotyps `xsdAnyContents`) angegeben sein.

Begründung:

Mit der Angabe des Namensraumes besteht die Möglichkeit, die übermittelten Inhalte, deren zugrunde liegende Schemata im Voraus nicht bekannt sind, einer Validierung zu unterziehen.

Prüfung:

Die Einhaltung dieser Empfehlung kann durch das XÖV-Produktionszubehör sichergestellt werden: XÖV-Invarianten.

NDR-8 (EMPFEHLUNG): Eindeutige versionsübergreifende Codes in Codelisten

a) Es wird empfohlen, Codes eine eindeutige versionsübergreifende Bezeichnung innerhalb der Codeliste zuzuweisen.

b) Es wird empfohlen, ungültig gewordene Codes nicht als Codes für andere Zwecke wiederzuverwenden.

Erläuterung:

Diese Regel bezieht sich auf Codes (Stereotyp `xsdCodeListEntry`).

Begründung:

Eindeutige Codes sind insbesondere im Kontext von Clearingstellen und Fachanwendungen von großer Bedeutung.

NDR-9 (EMPFEHLUNG): Umgang mit unscharfen Codelisten-Einträgen und nicht abgeschlossenen Codelisten

Für unscharfe Codelisten-Einträge und nicht abgeschlossene Codelisten wird empfohlen, diese auf eine spezifische Art und Weise im Standard zu modellieren.

Erläuterung:

Codelisten-Einträge werden als unscharf bezeichnet, wenn sie inhaltlich nicht ausreichend präzise sind. Sie sollten wie in Abschnitt 6.2.4 auf Seite 98 dargestellt im Standard umgesetzt werden. Codelisten, die nicht abgeschlossen sind, definieren nicht alle für einen bestimmten Anwendungskontext benötigten Werte als Codes. Sie sollten ebenfalls wie in Abschnitt 6.2.4 auf Seite 98 dargestellt im Standard abgebildet werden.

Begründung:

Die vorgeschlagenen speziellen Modellierungsmuster erlauben eine uneingeschränkte Handhabung der beiden oben beschriebenen Varianten von Codelisten. Eine einheitliche Verwendung dieser Muster erhöht die Interoperabilität und erleichtert die spätere Implementierung.

5.2.2 Namen

Namensregeln dienen der problemlosen Weiterverarbeitung der XML-Schemata im Zuge der Implementierung, helfen bei der Erstellung eines einheitlichen Standards und erleichtern die Integration externer Standards bzw. ermöglichen anderen Standards den eigenen Standard zu integrieren.

NDR-10 (MUSS): Konsistente Namen in UML-Modell und XML-Schemata

Die Namen von XML-Attributen, XML-Elementen und XML-Typen müssen mit den Namen der entsprechenden UML-Konstrukte (Attribute, Rollennamen und Klassen) übereinstimmen.

Begründung:

Die Bezeichnungen in einem UML-Modell müssen in jedem Fall eindeutig auf XML-Artefakte abgebildet werden. Eine abweichende Fachmodell-Semantik in verschiedenen XÖV-Standards führt zu Intransparenz und würde somit die Basis zur Erreichung von Interoperabilität auflösen.

Prüfung:

Die Einhaltung dieser Regel wird durch das XÖV-Produktionszubehör sichergestellt: XÖV-XSD-Vorlagen.

NDR-11 (SOLL): Erlaubte Zeichen für Namen

Namen von XML-Attributen, XML-Elementen und XML-Typen eines XÖV-Standards sollen nur Buchstaben, Ziffern, Punkte, Unterstriche und Bindestriche enthalten.

Erläuterung:

Namen von XML-Attributen (Stereotyp `xsdAttribute`), XML-Elementen (Stereotyp `xsdElement` oder `xsdGlobalElement`) und XML-Typen (Stereotyp `xsdNamedType`), die innerhalb eines XML-Schemas (Stereotyp `xsdSchema`) definiert sind, sollen nur die im Folgenden aufgeführten Zeichen beinhalten:

- `a-z` und `A-Z` (Buchstaben in Groß- und Kleinschreibung)
- `0-9` (Ziffern)
- `.` (Punkt)
- `_` (Unterstrich)
- `-` (Bindestrich)

Weitere Festlegungen zur Groß- und Kleinschreibung von (und in zusammengesetzten) Namen sind in NDR-6 auf Seite 66 zu finden.

Begründung:

Vor dem Hintergrund der Implementierbarkeit eines XÖV-Standards mit gängigen Technologien (Programmiersprachen, Bindings) müssen Einschränkungen bei der Nutzung von Namen berücksichtigt werden. Programmiersprachen akzeptieren im Allgemeinen keine Umlaute, Leerzeichen sowie andere Sonderzeichen innerhalb von Bezeichnern. Gängige XML-Binding-Werkzeuge akzeptieren zwar alle erlaubten XML-Namen, passen diese jedoch abhängig von den genutzten Sonderzeichen an, sodass die Sonderzeichen in der Regel nicht bestehen bleiben.

Hinweis: Die Verwendung der in dieser Regel zugelassenen Zeichen sollte hinsichtlich ihrer Eignung zur technischen Implementierung des XÖV-Standards durch das jeweilige XÖV-Vorhaben geprüft werden, da auch für diese eine Akzeptanz durch alle Binding-Werkzeuge nicht abschließend gesichert ist.

Prüfung:

Die Einhaltung dieser Regel kann durch das XÖV-Produktionszubehör sichergestellt werden: XÖV-Invarianten.

NDR-12 (SOLL): Erlaubte Zeichen für Klassifikationen in Namen

Zur Abbildung von Klassifikationen in Namen sollen Punkte verwendet werden.

Erläuterung:

Namen von XML-Attributen (Stereotyp `xsdAttribute`), XML-Elementen (Stereotyp `xsdElement` oder `xsdGlobalElement`) und XML-Typen (Stereotyp `xsdNamedType`), die innerhalb eines XML-Schemas (Stereotyp `xsdSchema`) definiert sind, sollen in ihrem Namen das Zeichen "." (Punkt) nur zur Abbildung einer Klassifikation verwenden.

Begründung:

Die Nutzung eines Punktes in Namen dient einem einheitlichen Klassifikationsmuster von XML-Schema-Bestandteilen und erleichtert bei der Betrachtung eines XÖV-Standards, die Zuordnung von zusammengehörenden Bestandteilen zu einer Klasse von XML-Schema-Bestandteilen.

Beispiel:

`String.Latin`, `hamsterzuchtregister.hamstergeburtsmeldung.0101`, `hamsterzuchtregister.hamstertodesmeldung.0103`, `CodeList.Hamstergattung`, `CodeList.Geschlecht`

Prüfung:

Die Einhaltung dieser Regel kann teilweise durch das XÖV-Produktionszubehör sichergestellt werden. Es wird geprüft, dass keine zwei Punkte aufeinander folgen: XÖV-Invarianten.

NDR-13 (SOLL): Eindeutige versionsübergreifende Namen von Nachrichten

a) Nachrichten sollen einen eindeutigen versionsübergreifenden Namen innerhalb eines Standards aufweisen.

b) Ungültige Nachrichtennamen sollen nicht für neue Nachrichten wiederverwendet werden.

Erläuterung:

Diese Regel bezieht sich auf Nachrichten (Stereotyp `xsdMessage`). Weitere Angaben zum Aufbau von Nachrichten-Namen mit Nachrichtennummern sind in NDR-17 auf Seite 71 zu finden.

Begründung:

Eindeutige Nachrichtennamen sind insbesondere im Kontext von Clearingstellen und Fachanwendungen von großer Bedeutung. Falls eine Nachricht im Rahmen der Fortschreibung des Standards entfällt, sollte ihr Name nicht wiederverwendet werden, damit es bei der Übertragung nicht zu Interpretationsproblemen kommen kann.

Beispiel:

`hamsterzuchtregister.hamstergeburtsmeldung.0101`

NDR-14 (EMPFEHLUNG): Namen in deutscher Sprache

Es wird empfohlen, alle Bestandteile eines XÖV-Standards in deutscher Sprache zu benennen.

Erläuterung:

Diese Regel bezieht sich auf XML-Attribute (Stereotyp `xsdAttribute`), -Elemente (Stereotyp

`xsdElement` oder `xsdGlobalElement`) und -Typen (`xsdNamedType`) sowie XML-Enumerations (Stereotyp `xsdCodeListEntry`).

Begründung:

Ein XÖV-Standard wird durch fachliche Anforderungen der öffentlichen Verwaltung Deutschlands bestimmt. Die Namen der Modellelemente spiegeln in der Regel diese Fachlichkeit wider. Im Falle von Standards im EU-Kontext oder bei technischen Begriffen kann jedoch die Wahl einer anderen Sprache geeignet sein. Auch in Standards der deutschen Verwaltung können sich Abweichungen ergeben. Insbesondere bei generischen Datentypen kann das der Fall sein, wenn diese zum Beispiel von W3C-Schema-Datentypen abgeleitet werden und die Herleitung im Namen des ableitenden Typs ausgedrückt werden soll.

NDR-15 (EMPFEHLUNG): Groß- und Kleinschreibung von (und in zusammengesetzten) Namen

a) Es wird empfohlen, in Namen von Typen eines XÖV-Standards alle eigenständigen Wörter mit einem Großbuchstaben zu beginnen (Upper Camel Case).

b) In den Namen von XML-Attributen und -Elementen wird eine analoge Schreibweise empfohlen. Hier sollte das erste Zeichen des Namens jedoch ein kleiner Buchstabe sein (Lower Camel Case).

c) Im Falle von Namen globaler XML-Elemente, und damit auch von Nachrichten, wird ebenfalls die Lower Camel Case Schreibweise empfohlen. Nach Klassifikationsmerkmalen in Namen von globalen Elementen sollte ebenfalls mit einem Kleinbuchstaben begonnen werden.

Erläuterung:

Diese Empfehlung bezieht sich auf XML-Typen (Stereotyp `xsdNamedType`), -Attribute (Stereotyp `xsdAttribute`) und -Elemente (Stereotyp `xsdElement` oder `xsdGlobalElement`). Weitere Festlegungen zu erlaubten Zeichen in Namen sind in NDR-11 auf Seite 68 sowie zur Namensstruktur von globalen Elementen in NDR-16 auf Seite 70 zu finden.

Beispiele:

zu a): (Typ) `Hamster`, `WettbewerbsartListeNichtAbgeschlossen`

zu b): (Element) `strasse`, `grundFuerAusfallDesWettbewerbs`
(Attribut) `listURI`

zu c): (Nachricht) `hamsterzuchtregister.hamstergeburtsmeldung.0101`

Prüfung:

Die Einhaltung der Empfehlung kann für alle Bestandteile nur eingeschränkt durch das XÖV-Produktionszubehör sichergestellt werden. Lediglich die Einhaltung für das erste Zeichen kann geprüft werden: XÖV-Invarianten.

NDR-16 (EMPFEHLUNG): Namensstruktur von globalen Elementen

a) Für Namen von globalen Elementen wird empfohlen, den Namen der Hauptgruppe als Präfix voranzustellen.

b) Der Name der Hauptgruppe als Präfix wird mit einem Punkt vom eigentlichen Namen des globalen Elements abgegrenzt.

Erläuterung:

Der Name der Hauptgruppe (siehe NDR-5 auf Seite 65) wird als Präfix für globale XML-Elemente verwendet (Stereotyp `xsdGlobalElement`). Die Angabe des eigentlichen Elementnamens erfolgt nach dem ersten Punkt. Weiterführende Empfehlungen zum Namensaufbau von Nachrichten (Stereotyp `xsdMessage`) als Spezialisierung eines globalen Elements sind in NDR-13 auf Seite 69 und NDR-17 auf Seite 71 sowie zum allgemeinen Vorgehen zur Groß- und Kleinschreibung in NDR-15 auf Seite 70 zu finden.

Begründung:

Die Angabe des Präfixes ermöglicht die Zuordnung von globalen Elementen zu einer sie umfassenden Hauptgruppe.

Beispiele:

- `hamsterzuchtregister.hamstergeburtsmeldung.0101` (`hamsterzuchtregister` = Hauptgruppe, `hamstergeburtsmeldung.0101` = Elementname)
- `hamsterzuchtregister.hamstertodesmeldung.0103` (`hamsterzuchtregister` = Hauptgruppe, `hamstertodesmeldung.0103` = Elementname)

Prüfung:

Die Einhaltung dieser Empfehlung kann durch das XÖV-Produktionszubehör sichergestellt werden: XÖV-Invarianten.

NDR-17 (EMPFEHLUNG): Eindeutige versionsübergreifende Nummern in Namen von Nachrichten

a) Es wird empfohlen, Nachrichten eine eindeutige versionsübergreifende Nummer innerhalb des Standards als Suffix im Namen zuzuweisen.

b) Das Suffix wird mit einem Punkt abgegrenzt.

c) Es wird empfohlen, ungültig gewordene Nachrichtennummern nicht für neue Nachrichten wiederzuverwenden.

Erläuterung:

Diese Regel bezieht sich auf Nachrichten (Stereotyp `xsdMessage`).

Begründung:

Eindeutige Nachrichtennummern sind insbesondere im Kontext von Clearingstellen und Fachanwendungen von großer Bedeutung.

Beispiel:

`hamsterzuchtregister.hamstergeburtsmeldung.0101`

Prüfung:

Die Einhaltung der Bestandteile a) und b) dieser Regel kann durch das XÖV-Produktionszubehör innerhalb innerhalb der aktuell vorliegenden Version des Standards, jedoch nicht versionsübergreifend sichergestellt werden: XÖV-Invarianten.

NDR-18 (EMPFEHLUNG): Namen von XML-Schema-Dateien

Es wird empfohlen, den Namen eines XÖV-Standards in den Namen der Schema-Dateien aufzuführen.

Erläuterung:

Diese Regel bezieht sich auf Dateinamen der XML-Schemata (Werte der Eigenschaft `schemaFile` des Stereotyps `xsdSchema`).

Begründung:

Die Zugehörigkeit der XML-Schemata zu ihrem Standard sollte auch auf Dateiebene direkt erkennbar sein.

Beispiel:

`xhamsterzucht-ehzr.xsd`

Prüfung:

Die Einhaltung dieser Empfehlung kann durch das XÖV-Produktionszubehör mit der Voraussetzung sichergestellt werden, dass der Name des "xsdXModel"-Modells zum XÖV-Standard dem Namen des XÖV-Standards entspricht: XÖV-Invarianten.

5.2.3 Dokumentation

Die Dokumentation der Bestandteile eines XÖV-Standards ist essentiell für seine Implementierung in Anbetracht des Umfangs und der Komplexität, den die Standards im Allgemeinen erreichen.

NDR-19 (SOLL): Dokumentation in deutscher Sprache

Es sollen alle Bestandteile eines XÖV-Standards in deutscher Sprache dokumentiert sein.

Erläuterung:

Neben der Dokumentation in deutscher Sprache können zusätzlich weitere Sprachen gewählt werden. Ein Abweichen von dieser Regel, d.h., keine Dokumentation in deutscher Sprache, sondern nur in anderen Sprachen, ist gegebenenfalls in anderen Kontexten sinnvoll.

Begründung:

Ein XÖV-Standard wird durch fachliche Anforderungen der öffentlichen Verwaltung Deutschlands bestimmt. Um die semantische Interoperabilität zwischen Standards der deutschen Verwaltung zu erhöhen, sollte die deutsche Sprache bei der Dokumentation der Modellelemente angewandt werden. Im Falle von Standards im EU-Kontext oder bei technischen Begriffen kann jedoch die Wahl einer anderen Sprache geeignet sein.

NDR-20 (EMPFEHLUNG): Dokumentation der Rechtsgrundlagen

Für Nachrichten eines XÖV-Standards wird empfohlen, die Rechtsgrundlagen innerhalb des Standards zu dokumentieren.

Erläuterung:

Die Rechtsgrundlagen (Wert der Eigenschaft `rechtsgrundlagen` des Stereotyps `xsdMessage`) zu einer Nachricht (Stereotyp `xsdMessage` und `xsdGlobalElement`) sollen dokumentiert werden.

Begründung:

Die Dokumentation der Rechtsgrundlagen dient der Nachvollziehbarkeit der rechtlichen Notwendigkeit für den Datenaustausch.

Prüfung:

Die Einhaltung dieser Empfehlung kann durch das XÖV-Produktionszubehör sichergestellt werden: XÖV-Invarianten.

NDR-21 (EMPFEHLUNG): Codenamen für Codelisten-Einträge

Es wird empfohlen, allen Einträgen (Codes) einer Codeliste ihren jeweiligen Codenamen zuzuweisen.

Erläuterung:

Diese Regel bezieht sich auf die Codes (Stereotyp `xsdCodeListEntry`) einer Codeliste. Einem Code sollte sein Codename zugewiesen werden (Wert der Eigenschaft `name` des Stereotyps `xsdCodeListEntry`).

Begründung:

Die einheitliche Beschreibung von Codenamen in einer Codeliste ermöglicht eine einheitliche Vorgehensweise bei der Implementierung der Codelisten. Unter anderem aus Gründen der historischen Nachvollziehbarkeit kann es erforderlich sein, in den XML-Instanz-Dokumenten neben dem Code auch den Codenamen zu übertragen.

Beispiel:

`Graue Zwerghamster` für den Code `GZH` in der Codeliste für `Codelist.Hamstergattung`

Prüfung:

Die Einhaltung dieser Empfehlung kann durch das XÖV-Produktionszubehör sichergestellt werden: XÖV-Invarianten.

5.2.4 Wiederverwendung

Die Wiederverwendung von existierenden fachlichen und technischen Bestandteilen des eigenen Standards, anderer XÖV-Standards sowie der von der XÖV-Koordination herausgegebenen Standards fördert die Einheitlichkeit innerhalb und zwischen Standards und damit deren Interoperabilität.

NDR-22 (MUSS): Unveränderte Übernahme von XÖV-Codelisten

Alle aus dem XRepository übernommenen Codelisten müssen hinsichtlich ihrer Codes und Codenamen sowie ihrer weiteren Eigenschaften unverändert im Standard abgebildet werden.

Erläuterung:

Falls eine versionsrelevante Codeliste, die im Standard modelliert wird, als eigenständige XÖV-Codeliste im XRepository existiert, muss diese in Hinblick auf die für XÖV relevanten Einträge genau dem XRepository-Pendant entsprechen (siehe auch Abschnitt 6.2.1 auf Seite 89).

Begründung:

Eine Verletzung dieser Regel hätte zur Folge, dass die Kommunikationspartner unterschiedliche Codelisten verwenden. Eine Interoperabilität bezüglich des Austauschs von Codes wäre in diesem Fall nicht mehr gewährleistet.

NDR-23 (MUSS): Umgang mit Restriktionen über unterschiedliche Namensräume

Für Restriktionen über Schemata mit unterschiedlichen Namensräumen müssen Attribute und lokale Elemente der Schemata in unqualifizierter Form definiert werden.

Erläuterung:

Grundsätzlich sollten alle Elemente in XML-Schema in qualifizierter Form (`form="qualified"`), d.h. in dem Namensraum des Schemas (`targetNamespace`), definiert werden, um Namenskonflikte zwischen gleichnamigen Elementen verschiedener Schemata zu vermeiden. Da es sich hierbei um den Standardfall handelt, kann die Angabe von `form` im UML-Modell entfallen, sie wird dann implizit als `qualified` angenommen. Für Restriktionen (Stereotyp `xsdRestriction`) über Namensraum-Grenzen hinweg müssen die lokalen XML-Elemente (Stereotyp `xsdElement`) allerdings davon abweichend im Default-Namensraum (form="unqualified") definiert werden. Dies gilt sowohl für den allgemeinen als auch für den speziellen Typ, der an der Restriktion teilnimmt.

Begründung:

XML-Schema erlaubt im Allgemeinen keine abweichenden Element-Namensräume bei Restriktionen.

Prüfung:

Die Einhaltung dieser Regel kann durch das XÖV-Produktionszubehör sichergestellt werden: XÖV-Invarianten.

NDR-24 (SOLL): Wiederverwendung generischer Nachrichten-Eigenschaften

Nachrichten eines XÖV-Standards bzw. deren Nachrichtenköpfe sollen von einem gemeinsamen Typen abgeleitet sein.

Erläuterung:

Alle Nachrichten (Stereotyp `xsdMessage` und `xsdGlobalElement`) bzw. ihre Köpfe werden von einem Typen, der generische Nachrichten-Eigenschaften umfasst, abgeleitet.

Begründung:

Die Ableitung aller Nachrichten bzw. ihrer Köpfe von einem Typen, der generische Nachrichten-Eigenschaften umfasst, vereinfacht die Implementierung des Standards durch die Systemhersteller.

Prüfung:

Die Einhaltung der Regel kann durch das XÖV-Produktionszubehör sichergestellt werden: XÖV-Invarianten.

NDR-25 (EMPFEHLUNG): Abgrenzung der Wiederverwendung durch Komposition und Restriktion

Es wird empfohlen, eine Komposition zu verwenden, um komplexere Strukturen mit einfachen, wiederverwendbaren "Bausteinen" eines Standards auszudrücken. Eine Restriktion sollte verwendet werden, um die "Bausteine" für die Wiederverwendung auf den spezifischen Sachverhalt anzupassen.

Erläuterung:

Für unterschiedliche Sachverhalte werden komplexe Datentypen mittels der Komposition wiederverwendbarer Bausteine z.B. aus dem "Baukasten" (siehe NDR-5 auf Seite 65) zusammengesetzt. Soweit möglich werden diese komplexen Typen durch Restriktion auf den spezifischen Sachverhalt so weit angepasst wie möglich.

Begründung:

Für die Qualität der übermittelten Daten und für die Umsetzbarkeit des Standards durch Fachverfahrenshersteller ist jeder Sachverhalt für sich so genau wie möglich mit Mitteln der XML-Schemata zu beschreiben, um den Interpretationsspielraum und damit die Fehleranfälligkeit zu minimieren. Hierfür ist die Restriktion grundsätzlich gut geeignet. Allerdings ist in der Modellierung und der Umsetzung eines Standards die Restriktion für mehrschichtige Restriktionen nicht sinnvoll handhabbar.

Negativbeispiel für das Ergebnis einer mehrschichtige Restriktion in einer XML-Instanz:

```
<xhz:hamster>
  <xhz:anschrift>
    <xhz:ort>Berlin</xhz:ort>
  </xhz:anschrift>
</xhz:hamster>
```

NDR-26 (EMPFEHLUNG): Physische Speicherorte von XML-Schemata als URL

Es wird empfohlen, den physischen Speicherort eines XML-Schemas in Form eines öffentlichen URLs anzugeben.

Erläuterung:

Diese Regel bezieht sich auf das XML-Schema-Attribut `schemaLocation` (Werte der Eigenschaft `schemaLocation` des Stereotyps `xsdSchema` bzw. die Kombination der Werteigenschaften von `schemaLocationBase` des Stereotyps `xsdXModel` und von `schemaFile` des Stereotyps `xsdSchema`).

Begründung:

XML-Schemata eines XÖV-Standards sollten öffentlich über einen URL zugänglich sein. Die Speicherorte werden bei der Einbindung von Schemata in andere Namensräume (mit Hilfe des XML-Schema-Elements `import`) genutzt, um den Schema-Validatoren mitzuteilen, wo die externen importierten XML-Schemata zu finden sind.

NDR-27 (EMPFEHLUNG): Verwendung von Original-Namespace-Präfixen bei Schema-Importen

Es wird empfohlen, Bestandteile eines importierten Schemas über ein Namensraum-Präfix anzusprechen, welches dem Original-Präfix des importierten Schemas entspricht.

Erläuterung:

Zur Einbindung externer XML-Schemata kann einerseits das originäre Namensraum-Präfix des importierten Schemas (Wert der Eigenschaft `prefix` des Stereotyps `xsdSchema` oder `xsdXModel` des externen Modells) oder ein anderes Präfix (durch Angabe eines Wertes für die Eigenschaft `prefix` des Stereotyps `xsdImport`) genutzt werden.

Begründung:

Aus Gründen der Transparenz und Nachvollziehbarkeit empfiehlt sich immer die Verwendung des originären Präfixes zu einem XML-Schema.

Beispiel:

Wenn Bestandteile des Standards XHamsterzucht in einem anderen Standard genutzt werden, sollten diese Elemente mit dem Präfix `xhz` (definiert über den Stereotyp `xsdSchema` oder `xsdXModel` in XHamsterzucht) angesprochen werden. Eine Umbenennung des Präfixes würde den Rückschluss auf den Ursprung der Bestandteile erschweren.

Prüfung:

Die Einhaltung dieser Regel kann durch das XÖV-Produktionszubehör sichergestellt werden: XÖV-Invarianten.

5.2.5 Technik und Infrastruktur

Die Regeln und Empfehlungen dieser Kategorie beziehen sich auf die technische Umsetzung eines XÖV-Standards in Form von XML-Schemata. Es werden Metadaten von XÖV-Schemata wie z. B. Version und Namensraum näher definiert.

NDR-28 (MUSS): Valide W3C-XML-Schemata

Alle XML-Schema-Dateien eines XÖV-Standards müssen gültig bezüglich der W3C-Schema-Spezifikation sein (siehe http://www.w3.org/2001/XMLSchema).

Begründung:

Ausschließlich valide XML-Schemata (Stereotyp `xsdSchema`) bilden eine Grundlage für den maschinellen Datenaustausch.

Prüfung:

Die Einhaltung dieser Regel wird durch das XÖV-Produktionszubehör sichergestellt: XÖV-XSD-Vorlagen und XÖV-Invarianten.

NDR-29 (MUSS): Identifizierende Namensräume

Die Namensräume der globalen XML-Elemente und benannten XML-Typen eines XÖV-Standards müssen den Standard eindeutig identifizieren. Jedes globale Element und jeder benannte Typ muss sich in einem entsprechenden Namensraum befinden und über ein Präfix angesprochen werden können.

Erläuterung:

Diese Regel bezieht sich auf die Ziel-Namensräume der XML-Schemata (Eigenschaft `namespace` des Stereotyps `xsdSchema` oder `xsdXModel`) sowie die dazugehörigen Präfixe der XML-Schemata (Eigenschaft `prefix` des Stereotyps `xsdSchema` oder `xsdXModel`).

Begründung:

Eindeutige Namensräume sind für die Wiederverwendung von Elementen und Typen eines Standards notwendig.

Prüfung:

Die Einhaltung dieser Regel wird durch das XÖV-Produktionszubehör sichergestellt: XÖV-XSD-Vorlagen und XÖV-Invarianten.

NDR-30 (MUSS): Versionierung der Schemata

Jedes Schema eines XÖV-Standards muss versioniert sein.

Erläuterung:

Die Regel bezieht sich auf das XML-Schema-Attribut `version` (Werte der Eigenschaft `version` des Stereotyps `xsdSchema` oder `xsdXModel`).

Begründung:

Versionen müssen Entwicklungsstände eines Schemas bzw. des XÖV-Standards eindeutig identifizieren. Dieses Vorgehen ist unter anderem für den Versionswechsel im Betrieb unentbehrlich.

Prüfung:

Die Einhaltung dieser Regel wird durch das XÖV-Produktionszubehör sichergestellt: XÖV-Invarianten.

NDR-31 (SOLL): Namensräume mit Versionen

Die in einem XÖV-Standard definierten Namensräume sollen die Version des Standards enthalten.

Erläuterung:

Diese Regel bezieht sich auf XML-Namensräume (Werte der Eigenschaft `namespace` des Stereotyps `xsdSchema` oder `xsdXModel`).

Begründung:

Eine neue Version eines XÖV-Standards definiert einen anderen Namensraum als dessen Vorgänger-Version. Bei der Wiederverwendung von Elementen eines XÖV-Standards, d.h. bei der Nutzung von Elementen eines bestimmten Namensraums, soll feststehen, welcher Version eines Standards die Elemente angehören.

6. Leitlinien zu Codelisten

Eine der größten Herausforderungen bei der Integration von IT-Fachverfahren ist die Herstellung semantischer Interoperabilität. Codelisten bieten zumindest für einen Teil des Problems eine sehr direkte Lösung. Sie definieren für viele zentrale Begriffe eine explizite und überprüfbare Semantik und sind durch IT-Verfahren mit verhältnismäßig geringem Aufwand umsetzbar. Aus diesen Gründen sollte bei der Entwicklung von XÖV-Standards die konsequente Verwendung von Codelisten überall da angestrebt werden, wo sie zur Klarstellung der Semantik von Begriffen im Datenaustausch beitragen kann.

Neben spezifischen Codelisten innerhalb der eigenen Fachlichkeit sind vor allem fachübergreifende bzw. fachunabhängige Codelisten, die in verschiedenen Kontexten wiederverwendet werden können, von ganz besonderer Bedeutung. Für den fachübergreifenden Datenaustausch wird es sich auszahlen, in deren Qualität zu investieren. Wesentliche Voraussetzung für einen breiten Einsatz solcher Codelisten ist jedoch deren zentrale Bereitstellung.

Folgende Beispiele sollen die Bedeutung fachunabhängiger Codelisten verdeutlichen: Für den Datenaustausch von IT-Verfahren der Bereiche Meldewesen, Ausländerwesen und Polizei wäre es hilfreich, über Typen von Ausweisdokumenten auf der Basis einer gemeinsamen Codeliste zu kommunizieren. Der Datenaustausch (und die anschließende interne Datenintegration) zu den Merkmalen Religionszugehörigkeit, Familienstand oder Staat zwischen Meldewesen, Personenstandswesen und der Steuerverwaltung von Bund und Ländern wird erleichtert, wenn gemeinsame Codelisten zu diesen Merkmalen vorliegen.

Die Erarbeitung solcher fachunabhängigen bzw. fachübergreifenden Codelisten wird durch die XÖV-Koordination entsprechend unterstützt. Die XÖV-Koordination moderiert, wo Bedarf besteht, unter Beteiligung der Interessenten die Abstimmung von Codelisten. Die Ergebnisse werden der Allgemeinheit an zentraler Stelle im XRepository unter www.xrepository.de zur Verfügung gestellt.

Im folgenden Abschnitt wird beschrieben, wie die zentrale Bereitstellung ausgestaltet ist und was diese für Herausgeber und Anwender der Codelisten bedeutet. Anschließend wird daraufeingegangen, wie Codelisten in XÖV-Standards eingebunden werden sollen (siehe Abschnitt 6.2 auf Seite 89).

6.1 Bereitstellung von Codelisten

Bisher werden von der öffentlichen Verwaltung genutzte Codelisten an unterschiedlichen Stellen, z. B. im Bundesanzeiger oder auf Websiten und in verschiedensten Formaten, z. B. als Access-Datenbank, XML-Datei oder PDF-Dokument veröffentlicht. Einen Überblick, welche Codelisten wo zur Verfügung stehen, gibt durch die XÖV-Koordination zumindest in Ansätzen. Dabei sind gerade die schnelle Auffindbarkeit und ein einheitliches, technisch zu verarbeitendes Format von Codelisten zwei entscheidende Kriterien für deren häufige Wiederverwendung.

6.1.1 Veröffentlichung und Dokumentation von Codelisten

In diesem Abschnitt wird darauf eingegangen, wie und mit welchen Informationen Codelisten im XRepository, der zentralen Plattform für XÖV-Inhalte, bereitgestellt werden sollen.

6.1.1.1 Zentral bereitgestellte Codelisten

Codelisten, die zum gewöhnlichen Veröffentlichungsumfang eines XÖV-Standards gehören, werden in der Regel nicht als eigenständige Objekte im XRepository veröffentlicht, sondern als Artefakte der einzelnen XÖV-Standards (enthalten in den XML-Schema-Dateien, der Dokumentation oder als extra Dokumente neben den XML-Schema-Dateien des XÖV-Standards). Es handelt sich üblicherweise um standardspezifische Codelisten.

Codelisten hingegen, die von fachübergreifendem Interesse sind, werden als eigenständige Objekte im XRepository veröffentlicht und dokumentiert.

Die meisten dieser Codelisten sind durch die XÖV-Koordination für die fachunabhängige Verwendung ausgewählt worden. Sie werden deshalb im XRepository mit dem Prädikat "Standard" gekennzeichnet. Zu nennen sind hier als Beispiele die Codeliste zum Thema "Staat und Staatsangehörigkeit", herausgegeben vom Statistischen Bundesamt ('Staatsangehörigkeits- und Gebietsschlüssel'), oder auch der bundeseinheitliche Schlüssel der Zugehörigkeit zu einer Religionsgemeinschaft (herausgegeben im Rahmen des DSMeld vom Bundesministerium des Innern). Die erwähnten Codelisten werden von einer Vielzahl von Stellen z.B. in Meldewesen, Personenstandswesen, Ausländerwesen, Justizwesen und Finanzwesen verwendet.

Hinweis: Die veröffentlichten Codelisten können durch einen XÖV-Standard auf eine frei gewählte Weise eingesetzt werden (versionsrelevant oder nicht, schemavalidierend oder nicht, siehe Abschnitt 6.2.3 auf Seite 91). Die Art der Verwendung wird nicht durch die Veröffentlichung im XRepository festgelegt.

Hinweis: Die zu veröffentlichenden Codelisten werden stets in einer Weise veröffentlicht, welche lizenzrechtlich vertretbar ist. Falls Urheberrechte zu berücksichtigen sind, welche die Weitergabe einer Codeliste einschränken, wird sie im jeweiligen XRepository-Eintrag lediglich beschrieben und referenziert, ihr Inhalt bzw. die Codeliste als Artefakt aber nicht veröffentlicht (siehe Abschnitt 6.1.3 auf Seite 88).

6.1.1.2 Codelisten bereitstellen – Anwendungsfälle

In diesem Abschnitt werden die für das Bereitstellen und die Pflege von Codelisten im XRepository notwendigen Akteure und ihre Aktivitäten vorgestellt.

Akteure

Folgende Akteure sind an dem Bereitstellungsprozess für Codelisten beteiligt:

Akteur	Zuständigkeitsbereich
Betreiber des XRepositorys	Betreiber des Dienstes XRepository allgemein. Gemeint ist hier nicht der technische Betrieb von Servern und Netzwerktopologie, sondern der Dienst, den das XRepository seinen Kunden anbietet. Dieser Dienst umfasst die fachliche Administration, den Support für die Anwender, sowie Änderungs- und Qualitätsmanagement einschließlich der Tätigkeiten von Moderator und Gutachter im Zuge der Pflege von Sammlung, Katalog und Standard (vgl. Pflegekonzept für das XRepository).
Herausgeber einer Codeliste	Dies ist die Stelle, welche die Verantwortung für die Codeliste hat, d. h. ihren Inhalt definiert und fortschreibt. Gemeint ist hier aber das Recht zum Definieren und Fortschreiben, weniger diese operativen Tätigkeiten selber.
Einsteller einer Codeliste	Dies ist die Stelle, die die Codeliste ins XRepository einstellt und dort die Ersteintragung der Metadaten vornimmt.
Pfleger einer Codeliste	Diese Stelle wartet den Eintrag der Codeliste im XRepository und schreibt ihn bei Bedarf fort.
Anwender einer Codeliste	Der Nutzer oder eine Anwendung, welche Informationen über eine Codeliste bzw. diese selber als Artefakt aus dem XRepository entnimmt und sie für bestimmte Zwecke z. B. in einem XÖV-Standard oder einem IT-Verfahren einsetzt.

Aktivitäten

Folgende Aktivitäten sind für die Bereitstellung einer fachunabhängigen Codeliste im XRepository auszuführen.

- **Nutzer benennen und freischalten für die Pflege einer zugewiesenen Codeliste.** Falls die Verantwortung für die Pflege wechselt, wirkt sich das auf die Berechtigungen von XRepository-Nutzern aus, welche entsprechend neu zuzuweisen sind. Um diese Neuzuweisung bei einem personellen Zuständigkeitswechsel innerhalb der verantwortlichen Organisation zu vermeiden, kann die Verwendung eines allgemeinen Nutzer-Accounts für das XRepository sinnvoll sein.
- **Codelisten transformieren.** Wenn die Codeliste dem Pfleger einer Codeliste nicht im passenden technischen Format zur Verfügung steht, wandelt er sie vor einer Veröffentlichung ggf. mittels technischer Hilfsmittel in das geeignete Format um.
- **Codelisten-Eintrag im XRepository anlegen.** Der Eintrag wird angelegt und über das entsprechend im XRepository angebotene Formular dokumentiert. Die Codeliste wird als Download-Datei eingehängt.
- **Codelisten-Eintrag im XRepository bearbeiten.** Die Dokumentation wird fortgeschrieben und ggf. die angehängte Codelisten-Datei gegen eine korrigierte oder fortgeschriebene Version ausgetauscht und der Versionszähler des XRepository-Eintrags hochgesetzt.
- **Support für Codelisten-Anwender anbieten.** Es wird Fragen und Probleme von Anwendern geben, die sich auf eine spezielle Codeliste beziehen. Entsprechend wird ein Dienst zur Unterstützung angeboten und ein Ansprechpartner (z. B. über eine E-Mail-Adresse) benannt.
- **Codelisten-Informationen abrufen.** Der Anwender recherchiert im XRepository Codelisten und informiert sich über Zweck und Inhalt von einzelnen recherchierten Codelisten.
- **Codelisten-Dateien abrufen.** Der Anwender legt die Codelisten-Datei per Download für die weitere Verwendung lokal ab.

6.1.1.3 Dokumentation von Codelisten im XRepository

Zu jeder Codeliste, die in das XRepository eingestellt wird, werden durch das XRepository Informationen zur Identität und weiteren Eigenschaften durch den Einsteller bzw. Pfleger einer Codeliste erhoben. Dabei ist der Eintrag der *obligatorischen* Metadaten Voraussetzung dafür, dass die Codeliste durch das XRepository abgespeichert werden kann. Die weitere Dokumentation der *optionalen* Metadaten, kann z. B. zu einem späteren Zeitpunkt eingepflegt werden. In diesem Abschnitt werden die Pflichtfelder und optionalen Metadaten für Codelisten aufgeführt und kurz erläutert.

Pflichtfelder

Die folgenden Metadaten sind als Dokumentation zu jeder Codeliste im XRepository als Pflichtfelder anzugeben:

Metadatum	Beschreibung
Name / Titel	Der Name der Codeliste, wie er vom Herausgeber der Codeliste vergeben wurde.
URI	Um die Codeliste versionsübergreifend zu identifizieren, wird eine URI verwendet, welche ebenfalls vom Herausgeber der Codeliste zu vergeben ist. Zu verwenden ist die URN-Syntax des W3C-Datentyps anyURI (Beispiel: "urn:oasis:names:specification:docbook:dtd:xml:4.1.2").
Version	Die Versionsnummer der in diesem Eintrag eingestellten Codeliste. Format und Aussage der Versionsnummer folgt dem Pflegekonzept des Herausgebers der Codeliste. Versionsnummern für Codelisten sind im XRepository auf maximal 16 Zeichen begrenzt.
Herausgeber	Dies ist die Stelle oder Organisation, die für die Codeliste verantwortlich ist in dem Sinne, dass sie über Änderungen der Codeliste entscheiden kann.
Beschreibung	Charakterisiert die Semantik, die durch die Codeliste definiert wird. Die Beschreibung drückt aus, welchen Begriff die Codeliste definiert (z. B. den Begriff "Familienstand"), in welchem Kontext die Codeliste anzuwenden ist bzw. für welchen Zweck sie definiert ist.
gültig ab (Datum)	Vom Herausgeber der Liste ist ein Datum zu setzen, ab dem die vorliegende Version der Codeliste aus seiner Sicht *wirksam* ist.

Optionale Felder

Ergänzend zu den verpflichtenden Eigenschaften sollen, soweit zutreffend, die folgenden Zusatzinformationen zu einer Codeliste im XRepository erfasst werden.

Metadatum	Beschreibung
Einsteller	Der XRepository-Nutzer, der die Codeliste eingestellt hat.
Pflegende Stelle	Die Organisation, welche die Fortschreibung der Codeliste im XRepository operativ durchführt, also dafür sorgt, dass z. B. aktuelle Versionen verfügbar sind.
Kontakt	Ansprechstelle für Fragen und Probleme zu einer Codeliste. Sollte aus einer Bezeichnung und einer E-Mail-Adresse bestehen sowie ggf. zusätzlich einer Telefonnummer.
Verwendung	Angabe, in welchen XÖV-Standards die Codeliste genutzt wird.

Metadatum	Beschreibung
Lizenz	Bezeichnung und Art der Lizenz, unter der die Codeliste im XRepository veröffentlicht ist. Verweist auf eine Dokumentation der Lizenz, welche ggf. im Download-Bereich zur Codeliste zur Verfügung gestellt wird.
Aktualisierung	Angabe, wie oft mit Änderungen an der Codeliste zu rechnen ist (Verfügbarkeit von Folgeversionen). Ggf. kann diese Information aus dem Pflegekonzept der jeweiligen Codeliste entnommen werden.
Aktualitätszusicherung	Art der Pflege, die für diese Codeliste vorgesehen ist: Zusicherung, dass sie im XRepository stets in der aktuellen Version angeboten wird. Wenn diese Zusicherung nicht gegeben wird, bedeutet dies, dass hier nur allgemeine Informationen zur Codeliste angegeben werden.
XRepository-Status der Codeliste	Sammlung, Katalog oder Standard (vgl. Hilfe zum XRepository https://www.xrepository.deutschland-online.de/xrepository/help.xhtml).
Ort der Ressource	Bezeichnet den Ort der Ressource, die die Codeliste für den Zugriff durch Anwendungen repräsentiert.(Noch nicht im XRepository umgesetzt.)
Format	Formatangabe (wie z.B. CSV, MDB, XML, PDF) insbesondere für Codelisten, die aus lizenzrechtlichen Gründen nicht im XRepository veröffentlicht, sondern nur mit ihren Metadaten dort gepflegt werden. Formate können auch versionsabhängig sein. Das Format der im XRepository veröffentlichten Codeliste ist OASIS Genericode, und muss hier nicht angegeben werden.

Darüber hinaus bietet das XRepository weitere allgemeine Metadaten:

- Projekt: Im Falle einer fachspezifischen Codeliste kann hier der zugehörige XÖV-Standard eingegeben werden.
- Auftraggeber: Stelle, die die Erstellung und Fortschreibung der Codeliste beauftragt.
- Stichwörter: Alle Begriffe, unter denen die Codeliste im XRespository gefunden werden soll, die nicht als separte Metadaten angeboten werden. Hier können z.B. auch einige der wichtigsten Einträge der Codeliste eingetragen werden.

6.1.1.4 Verantwortung für die Pflege der Codelisten im XRepository

Definition: Umfang der Pflege im XRepository

Die Pflege einer Codeliste im XRepository umfasst das Überwachen von durch den Herausgeber genehmigten Änderungen an der Codeliste durch die pflegende Stelle sowie das Veröffentlichen dieser Änderungen als eine **neue** Version im XRepository.

Die Pflege einer Codeliste im XRepository umfasst **nicht** die Prozesse zur operativen Pflege der Codeliste durch den Herausgeber der Codeliste selbst.

Pflegende Stellen für Codelisten

Fachspezifische Codelisten werden vom XÖV-Vorhaben gepflegt und im Rahmen der Veröffentlichung von neuen Ergebnissen wie z.B. einer neuen XÖV-Standard-Version im XRepository veröffentlicht.

Codelisten, die nicht durch XÖV-Vorhaben erstellt werden, sollten durch dessen Herausgeber im XRepository zur Verfügung gestellt werden.

Beispiele für fachunabhängige Codelisten und ihre Verantwortlichen

Nach der Planung der XÖV-Koordination sollen folgende Codelisten im XRepository verfügbar gemacht werden, sobald die Rahmenbedingungen dies zulassen:

- Staatsangehörigkeits- und Gebietsschlüssel. Herausgeber: Statistisches Bundesamt
- Familienstand (DSMeld, XMeld, XPersonenstand). Herausgeber: BMI / AG DSMeld
- Gerichtskennzahlen. Herausgeber: BMJ oder XJustiz
- Religionsschlüssel (Meldewesen, Personenstandswesen). Herausgeber: BMI
- Steuererhebende Religionsgemeinschaften. Herausgeber: FM NRW

6.1.2 Technisches Format für die Veröffentlichung von Codelisten

6.1.2.1 Zweck des Formats

Das technische Format OASIS Genericode (siehe Abschnitt 6.1.2.2 auf Seite 84), welches für die zentrale Bereitstellung von fachunabhängigen bzw. fachübergreifenden Codelisten vorgesehen ist, dient als universaler Standard für die Darstellung und den Austausch von Codelisten. Alle Codelisten im XRepository werden in diesem Format verfügbar gemacht. Zusätzlich können weitere Formate einer Codeliste angeboten bzw. als Metadatum dokumentiert werden.

Alle benötigten Informationen zu einer Codeliste lassen sich auf der Basis des technischen Formats für die zentrale Bereitstellung darstellen:

- Zunächst sind dies alle relevanten **Metadaten zu einer Codeliste**. Sie geben z. B. Auskunft über mögliche Kontexte, Herausgeber, eindeutige Identifizierung. Diese Informationen werden zu einem bestimmten Anteil redundant innerhalb der Repräsentation der Codeliste selber und in der XRepository-Dokumentation zur Codeliste geführt. Die Angabe der Informationen ist für beide Sachverhalte notwendig.
- Hinzu kommen die **Einträge der Codeliste**. Sie bilden den Inhalt der Liste. Zu jedem dieser Einträge werden der Code, dessen Bedeutung und ggf. weitere Informationen dargestellt.

Die technischen Eigenschaften des Formats machen es möglich, die bereitgestellten Codelisten für medienbruchfreien Austausch, Verarbeitung und Weiterverarbeitung durch Anwendungen einzusetzen.

Das universale Format zur Veröffentlichung und zum Austausch von Codelisten ist zu unterscheiden vom Format für die XML-Schema-Validierung von Codelisten. Die XML-Schema-Validierung ist lediglich eine spezielle Anwendung einer Codeliste in einem XÖV-Standard (siehe Abschnitt 6.2.3 auf Seite 91).

Die allgemeinen Anforderungen an das technische Format werden durch die Darstellung von Codelisten als XML-Instanzen optimal erfüllt, denn XML ist als universales Format für die Darstellung von Objekten sowie zum plattformübergreifenden Austausch und zur Verarbeitung von Daten allgemein etabliert. Außerdem lassen sich Codelisten als XML-Instanzen leicht in der Realisierung von IT-Anwendungen für die diversen Anwendungsbereiche, z. B. Bedienoberfläche, Strukturierung von Fachlogik, Datenaustausch mit externen Diensten einsetzen.

6.1.2.2 XML-Format OASIS Genericode zur Veröffentlichung von Codelisten im XRepository

Für die zentrale Bereitstellung von Codelisten wird in XÖV das XML-Format *OASIS Genericode* verwendet. Es ist ein Standardisierungsergebnis des *Code List Representation Technical Committee* (TC) der internationalen Standardisierungsorganisation OASIS[1]. OASIS Genericode stellt Codelisten als XML-Instanzen dar. Das Format geht auf vielfältige Anforderungen der Repräsentation von Codelisten ein, ge-

1.siehe http://docs.oasis-open.org/codelist/cs-genericode-1.0/doc/oasis-code-list-representation-genericode.html

stattet es aber auch, einfache Codelisten auf einfache Weise darzustellen. Die Bezeichnung Genericode bedeutet so viel wie *generisch einsetzbares Format für Codelisten*. Als Abkürzung hat sich "gc" durchgesetzt.

Nach der Genericode-Konzeption wird eine Codeliste als mehrspaltige Tabelle aufgefasst. Eine Codeliste besteht danach aus *Einträgen* (entry), von denen jeder einer *Zeile* (row) der Tabelle entspricht. Eine Codeliste für die Wochentage würde somit sieben Einträge vorsehen: je eine Zeile pro Wochentag. Die Tabelle kann nach gc aus beliebig vielen *Spalten* (column) bestehen. Jede Spalte gibt eine Metainformation zum Eintrag (z. B. 'Name des Wochentags auf französisch' oder 'Rolle des Tages in der Kirchenwoche (Werktag, Festtag)'), die grundsätzlich alle gleichberechtigte Aspekte der Codeliste sind. Eine Spalte, die zur Identifikation eines Eintrags der Codeliste geeignet ist, wird als Schlüssel ('key') gekennzeichnet und ist damit zur Laufzeit als 'code' einsetzbar. Mehrere Spalten oder auch Spaltenkombinationen können als Schlüssel wirken.

Vorteil der Entscheidung für Genericode ist die Verwendung von Ergebnissen aus internationalen Standardisierungsaktivitäten. Es ist somit gewährleistet, dass die XÖV-Koordination bei Bedarf über die OASIS TC Einfluss auf die Weiterentwicklung von Genericode nehmen kann.

Es ist angedacht, zukünftig weitere Funktionalitäten in das Genericode-Format zu integrieren. In der Diskussion sind z. B. die Darstellung von Teilmengen-Ganzes-Beziehungen zwischen Codelisten sowie die Aufnahme von diachronischen Informationen. (Die Zuordnung, welchem aktuellen Wert ein bestimmter historischer Eintrag der Codeliste entspricht, ist z. B. im Zusammenhang mit der Rechtsnachfolge von Behörden interessant.) Zusammenhänge und Abhängigkeiten von Codelisten untereinander (z. B. Religionsschlüssel und steuererhebende Religionsgemeinschaften), die vorläufig mit Mitteln des XRepositorys dargestellt werden, sollen langfristig auch mit Genericode abbildbar sein.

Genericode bietet vielfältige Möglichkeiten bei der Darstellung von Codelisten. Folgende Anforderungen existieren für die Darstellung von XÖV-Codelisten in Genericode:

1. Immer genau *eine* Spalte zu einer Codeliste ist als 'code' für den XÖV-Datenaustausch festgelegt (entspricht auf diese Weise dem Attribut "code" im XÖV-Basisdatentyp Code, siehe Abschnitt 4.3.2.2 auf Seite 57).
2. Zusätzliche Spalten sind für weitere Informationen zu einem Eintrag der Codeliste möglich, welche in IT-Verfahren nützlich sein können. Beispiele: AGS zur Justizbehörde; "gültig ab" für einen Staatencode.

Welche Spalte dem Attribut "name" im XÖV-Basisdatentyp Code (siehe Abschnitt 4.3.2.2 auf Seite 57) entspricht, muss bei Bedarf im XÖV-Standard festgelegt werden.

Die Verwendung des Formats Genericode in XÖV soll an einem fiktiven Beispiel demonstriert werden. Das Amt für Hamsterzucht ist Herausgeber einer Codeliste für Hamsterzucht-Wettbewerbspreise, aktuelle Version 1.0. Die Liste umfasst sechs verschiedene Preise, also sechs Einträge sowie eine Spalte mit einer Zusatzinformation über die regionale Beschränkung der Preisverleihung. Damit gibt es drei Spalten zu Schlüssel, Prämierung und Region. Im Format OASIS Genericode werden diese Eigenschaften nach den oben genannten Regeln durch die folgenden XML-Fragmente dargestellt.

In einem einleitenden Abschnitt wird Definition und Gültigkeit der Liste benannt:

```
<Annotation>
  <Description>
    <xoev-cl:beschreibung>Die Prämien, die ein Hamster erhalten kann.</xoev-cl:beschreibung>
     <xoev-cl:datumgueltigkeitab>2006-01-01</xoev-cl:datumgueltigkeitab>
  </Description>
```

```
  </Annotation>
```

Dann wird über weitere identifizierende Parameter, u.a. die Versionierung und den Herausgeber informiert:

```
<Identification>
  <ShortName>Hamsterzuchtprämien</ShortName>
  <LongName>Hamsterzuchtprämien der deutschen Hamsterzuchtliga</LongName>
  <Version>1.0</Version>
  <CanonicalUri>urn:de:xoev:xhamsterzucht:hamsterpraemierung</CanonicalUri>
  <CanonicalVersionUri>urn:de:xoev:xhamsterzucht:hamsterpraemierung-1.0</CanonicalVersion-
Uri>
  <Agency>
    <ShortName>BuHaz</ShortName>
    <LongName>Bundesamt für Hamsterzucht</LongName>
  </Agency>
</Identification>
```

Im darauf folgenden Abschnitt werden die Spalten ("Column") definiert, welche die Codeliste bilden. Dies sind die Spalten "Schluessel" (Pflichtspalte), "Prämierung" (Pflichtspalte) und "Region" (optionale Spalte). Die Spalte "Schluessel" wird für XÖV-Zwecke als CodeKey ('key') ausgezeichnet.

```
<ColumnSet>
  <Column Id="Schluessel" Use="required">
    <ShortName>Schluessel</ShortName>
    <Data Type="string"/>
  </Column>
  <Column Id="Prämierung" Use="required">
    <ShortName>Prämierung</ShortName>
    <Data Type="string"/>
  </Column>
  <Column Id="Region" Use="optional">
    <ShortName>Region</ShortName>
    <Data Type="string"/>
  </Column>
  <Key Id="codeKey">
    <Annotation>
      <AppInfo>
        <xoev-cl:xoev-code/>
      </AppInfo>
    </Annotation>
    <ShortName>CodeKey</ShortName>
    <ColumnRef Ref="Schluessel"/>
  </Key>
</ColumnSet>
```

Schließlich folgen die inhaltlichen Einträge der Liste. Für jeden Eintrag ("Row") existieren Werte für die beiden Pflichtspalten und in einigen Fällen auch Werte für die optionale Spalte.

```
<SimpleCodeList>
  <Row>
    <Value ColumnRef="Schluessel">
      <SimpleValue>001</SimpleValue>
    </Value>
    <Value ColumnRef="Prämierung">
      <SimpleValue>Deutscher Hamsterzuchtpreis</SimpleValue>
    </Value>
  </Row>
  <Row>
    <Value ColumnRef="Schluessel">
      <SimpleValue>002</SimpleValue>
    </Value>
    <Value ColumnRef="Prämierung">
      <SimpleValue>Elite-Hamster der Deutschen Hamsterzuchtvereine</SimpleValue>
    </Value>
  </Row>
  <Row>
    <Value ColumnRef="Schluessel">
      <SimpleValue>003</SimpleValue>
```

```
      </Value>
      <Value ColumnRef="Prämierung">
        <SimpleValue>Große Münchener Staatsprämie der Hamsterzucht</SimpleValue>
      </Value>
      <Value ColumnRef="Region">
        <SimpleValue>Bayern</SimpleValue>
      </Value>
    </Row>
    <Row>
      <Value ColumnRef="Schluessel">
        <SimpleValue>004</SimpleValue>
      </Value>
      <Value ColumnRef="Prämierung">
        <SimpleValue>Kleine Münchener Staatsprämie der Hamsterzucht</SimpleValue>
      </Value>
      <Value ColumnRef="Region">
        <SimpleValue>Bayern</SimpleValue>
      </Value>
    </Row>
    <Row>
      <Value ColumnRef="Schluessel">
        <SimpleValue>005</SimpleValue>
      </Value>
      <Value ColumnRef="Prämierung">
        <SimpleValue>Bremer Staatsprämie der Hamsterzucht</SimpleValue>
      </Value>
      <Value ColumnRef="Region">
        <SimpleValue>Hansestadt Bremen</SimpleValue>
      </Value>
    </Row>
    <Row>
      <Value ColumnRef="Schluessel">
        <SimpleValue>006</SimpleValue>
      </Value>
      <Value ColumnRef="Prämierung">
        <SimpleValue>Frankfurter Leistungshamster</SimpleValue>
      </Value>
      <Value ColumnRef="Region">
        <SimpleValue>Hessen</SimpleValue>
      </Value>
    </Row>
  </SimpleCodeList>
```

6.1.2.3 Repräsentation der XÖV-Metadaten einer Codeliste in OASIS Genericode

Die nachfolgende Tabelle listet die in XÖV benötigten Metadaten zu einer Codeliste (siehe Abschnitt 6.1.1.3 auf Seite 82) auf und gibt an, wo die entsprechenden Informationen in einer Genericode-XML-Datei abzulegen bzw. dort bei Bedarf aufzufinden sind. Im Falle von Metadaten, welche nur für die Dokumentation des XRepository-Eintrags vorgesehen sind, aber nicht für die Dokumentation in einer gc-Datei, steht "n. z." für "nicht zutreffend".

Metadatum[1]	**Genericode**
Name / Titel*	/gc:CodeList/Identification/ShortName, /gc:CodeList/Identification/LongName
URI *	/gc:CodeList/Identification/CanonicalUri
Version*	/gc:CodeList/Identification/Version, /gc:CodeList/Identification/CanonicalVersionUri
Herausgeber*	/gc:CodeList/Identification/Agency/ShortName, /gc:CodeList/Identification/Agency/LongName, /gc:CodeList/Identification/Agency/Identifier

Metadatum[1]	Genericode
Beschreibung*	/gc:CodeList/Annotation/Description
gültig ab (Datum)*	/gc:CodeList/Annotation/Description bietet hierfür Raum. Es können Elemente definiert werden, die das Datum der Gültigkeit aufnehmen können.
Ort der Ressource	/gc:CodeList/Identification/LocationUri
Format	n. z.
Einsteller	n. z.
Pflegende Stelle	n. z.
Kontakt	/gc:CodeList/Annotation/Description falls sinnvoll, sonst: n. z.
Anwendung	n. z.
Lizenz	n. z.
Aktualisierung	n. z.
Aktualitätszusicherung	n. z.
XRepository-Status der Codeliste	n. z.

1. Mit * gekennzeichnete Metadaten sind Pflichtfelder im XRepository.

6.1.2.4 XÖV-Werkzeugunterstützung für die Verarbeitung von Codelisten im Format OASIS Genericode

Codelisten sollen für XÖV im XRepository im Format OASIS Genericode bereitgestellt werden. Das stellt die Herausgeber von Codelisten vor einige Anforderungen. Sie müssen ihre Codelisten nicht nur einmalig nach OASIS Genericode überführen und dann veröffentlichen, sondern sie müssen sich außerdem um die Pflege ihrer XRepository-Einträge kümmern. Auch Folgeversionen müssen nach Genericode konvertiert und veröffentlicht werden.

Um die Bereistellung von Codelisten in diesem Format zu vereinfachen, hat die XÖV-Koordination das Werkzeug *Genericoder*[1] entwickelt und kostenfrei auf der Basis einer Open Source-Lizensierung zur Verfügung gestellt. Der Genericoder ist ein generisches Werkzeug, welches Codelisten in Form einer CSV-Datei einliest und auf der Basis konfigurierbarer Parameter in eine Genericode-XML-Datei transformiert.

6.1.3 Rechtliche Bedingungen zur Veröffentlichung von Codelisten im XRepository

Es ist durch den Herausgeber zu kommunizieren, unter welcher Lizenz eine Codeliste im XRepository veröffentlicht wird.

In Bezug auf Codelisten aus der öffentlichen Verwaltung ist eine entsprechende Entscheidung im Einzelfall zu treffen. Das allgemeine Ziel ist, Codelisten möglichst öffentlich und kostenfrei zur Verfügung zu stellen, wie es auch mit sonstigen Produkten der XÖV-Standardisierung geschieht.

1.siehe Joinup-Plattform unter https://joinup.ec.europa.eu/software/genericoder/home

Neben der Lizenz ist für Anwender die Aktualität einer veröffentlichten Codeliste von Bedeutung. Von dieser Informationspflicht können sowohl der Betreiber des XRepository, der Einsteller der Codeliste als auch der Herausgeber der Codeliste betroffen sein. Haftungsfragen (dass z.B. der Dienstbetreiber für die Korrektheit einer veröffentlichten Codeliste einsteht) sind hier weniger relevant, denn sie lassen sich per Erklärung ausschließen.

Für gewisse Kontexte ist auch eine Zusicherung von Aktualität anzustreben. Dies ist für bestimmte Codelisten im Sinne der Dienstqualität wesentlich.

Das folgende Szenario wird durch die XÖV-Koordination empfohlen:

1. Ein Herausgeber hat die Aufgabe übernommen eine bestimmte Codeliste im XRepository bereitzustellen.
2. Dies geschieht auf der Basis einer Absprache (ggf. unter Haftungsauschluss), dass er stets dafür sorgt, dass die Codeliste fehlerfrei in der jeweils aktuellen Version verfügbar ist.
3. Der Nutzer des Dienstes wird in der Dokumentation bzw.über die XRepository-Metadaten "Aktualitätszusicherung" und "Aktualisierung" (siehe Seite 82) der betreffenden Codeliste darüber informiert.

Mit Herausgebern von Codelisten der öffentlichen Verwaltung wird verbindlich abgesprochen, wie die Codelisten in diesem Sinne zu pflegen sind.

6.2 Anwendung von Codelisten in XÖV-Standards

Nachdem geklärt ist, wie Codelisten zentral bereitgestellt werden bzw. auffindbar sind, wird in diesem Abschnitt auf die konkrete Darstellung und die verschiedenen Möglichkeiten zur Einbindung von Codelisten in einen XÖV-Standards eingegangen.

6.2.1 Versionsrelevanz und Schemavalidierung

Bevor mit der Modellierung zur Einbindung von Codeliste im XÖV-UML-Modell begonnen werden kann, muss geklärt sein, ob eine Codeliste versionsrelevant in einen Standard eingebunden werden soll und ob ihre Werte bei der Schemavalidierung Berücksichtigung finden. Im Folgenden werden die Begriffe zu diesen zwei wichtigen Aspekte erläutert.

6.2.1.1 Versionsrelevanz

Eine Codeliste kann in einem XÖV-Standard versionsrelevant oder nicht-versionsrelevant integriert werden.

- **versionsrelevant**: Sowohl der URI als auch die Version der Codeliste werden in dem Datentyp des XÖV-Standards, der die Codeliste verwendet, festgelegt. Die konkreten Codes der Codeliste können, müssen aber nicht, mit dem XÖV-Standard herausgegeben – also in seinem XML-Schema oder seiner Dokumentation aufgelistet – werden. Die versionsrelevante Verwendung einer Codeliste bindet also immer eine definierte Version einer bestimmten Liste an eine definierte Version eines XÖV-Standards.
- **nicht-versionsrelevant**: Auch hier sind zwei Möglichkeiten zu unterscheiden. Im ersten Fall wird im Code-Datentyp die Identität der Codeliste durch einen URI festgelegt, aber die zugehörige Version wird offen gelassen. Im zweiten Fall wird, abgesehen von der groben Festlegung zur fachlichen Ausrichtung (z.B. Staaten), nicht festgelegt, welche konkrete Codeliste zu verwenden ist. Es wird somit

weder ein URI noch eine Version zur Codeliste eingetragen. In beiden Fällen werden die eigentlichen Codes der Codeliste nicht mit dem XÖV-Standard herausgegeben, da die konkrete Liste, auf die sich eine Nachricht des XÖV Standards bezieht, nicht eindeutig über URI und Version benannt ist. Für nicht-versionsrelevante Codelisten ist im Zuge des jeweiligen Nachrichtenaustauschs zur Laufzeit durch die Anwendung in der XML-Instanz der Nachricht festzulegen, welche Version und ggf. sogar welche konkrete Codeliste (URI) gemeint ist.

6.2.1.2 Validierung per XML-Schema

Eine Codeliste, die durch einen XÖV-Standard eingebunden wird, kann auf verschiedene Weise für die Prüfung von Nachrichten als XML-Instanzen verwendet werden. Hierbei ist zwischen schemavalidierenden und nicht-schemavalidierenden Codelisten zu unterscheiden:

- **schemavalidierend**: Wird eine Codeliste in einem XÖV-Standard versionsrelevant mit ihren konkreten Codes eingebunden, dann kann die Codeliste über einen spezifischen Code-Datentyp (siehe Typ 1, Abschnitt 6.2.3 auf Seite 91) mit allen ihren Codes in die XML-Schema-Dateien aufgenommen werden. Die XML-Schema-Dateien des XÖV-Standards prüfen somit auch die Inhalte von Code-Elementen der Nachrichten-XML-Instanzen. Ist eine Nachricht schemakonform, so ist sichergestellt, dass in der Nachricht nur Codes verwendet werden, die in den schemavalidierenden Codelisten definiert sind.
- **nicht-schemavalidierend**: Ist eine Codeliste nicht versionsrelevant und/oder die konkreten Codes nicht im XÖV-Standard hinterlegt, dann lässt sich das XML-Schema nicht für die Prüfung von Codes in Nachrichten-XML-Instanzen einsetzen. Die übertragenen Codes in den Nachrichten werden durch die beteiligten IT-Anwendungen dann an anderer Stelle und unter Nutzung anderer Technologien geprüft.

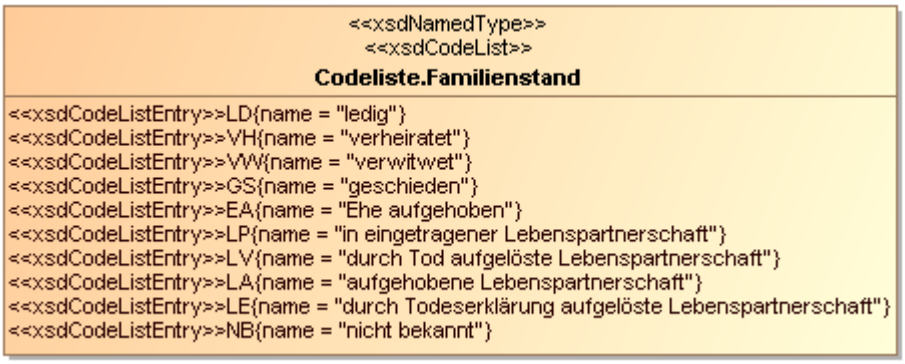
Codelisten für die Schemavalidierung einzusetzen, hat eine sehr strikte Kontrolle des Nachrichtenaustauschs mit einfachen und nachvollziehbaren Mitteln zur Folge. Das ist in vielen Fällen ein Vorteil. Es können aber auch Anforderungen vorliegen, dass die Prüfung der Standardkonformität der Nachrichten liberaler erfolgen soll, als dies durch eine Schemavalidierung geschehen würde. Insbesondere kann gewollt sein, dass nichtvorgesehene Codes in einer Nachricht nicht zur Abweisung der Nachricht wegen XML-Schema-Konflikten führen soll, sondern differenziertere Konsequenzen einzuleiten sind. Für solche bekannten Fälle sollten die Codes trotzdem konkret im Standard definiert werden und die nicht abgedeckten Codes über spezifische Modellierungsmuster im XÖV-Standard realisiert werden (siehe Abschnitt 6.2.4 auf Seite 98).

6.2.2 Darstellung der Einträge von Codelisten im XÖV-Standard

Wenn die Einträge einer Codeliste mit einem XÖV-Standard ausgeliefert werden sollen, so werden die Einträge (die Codes und ihre Bedeutung) der Codeliste im XÖV-UML-Modell des XÖV-Standards als Literale einer UML-Enumeration dargestellt. Die Enumeration ist dann mit dem Stereotyp `xsdCodeList` (Abschnitt 4.2.5 auf Seite 31) zu kennzeichnen, jedes der Literale wird durch den Stereotyp `xsdCodeListEntry`" (Abschnitt 4.2.6 auf Seite 32) charakterisiert.

Bild 6-1 Verwendung der Stereotypen xsdCodeList und xsdCodeListEntry für Codelisten und

ihre Codes



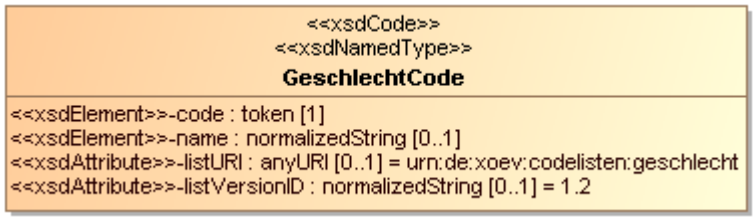
Wenn ein schemavalidierender Einsatz der Codeliste beabsichtigt ist, dann werden die Codes der Codeliste aus dem XÖV-UML-Modell mittels XÖV-Produktionszubehörs in Konstrukte der Art `xs:enumeration` in die XML-Schema-Dateien des XÖV-Standards übertragen:

```
<xs:simpleType>
  <xs:restriction base="xs:token">
    <xs:enumeration value="LD"/>
    <xs:enumeration value="VH"/>
    <xs:enumeration value="VW"/>
    <xs:enumeration value="GS"/>
    <xs:enumeration value="EA"/>
    <xs:enumeration value="LP"/>
    <xs:enumeration value="LV"/>
    <xs:enumeration value="LA"/>
    <xs:enumeration value="LE"/>
    <xs:enumeration value="NB"/>
  </xs:restriction>
</xs:simpleType>
```

6.2.3 Einbindung von Codelisten in XÖV-Standards – Code-Datentypen

Um eine Codeliste (z. B. die Codeliste Geschlecht) in einem XÖV-Standard einzusetzen, muss sie in die XML-Schema-Darstellung des Standards eingebunden werden. Für XÖV-Standards erfolgt die Einbindung der Codelisten einheitlich auf der Grundlage des XÖV-Basisdatentyps Code (Abschnitt 4.3.2 auf Seite 56). Für die unterschiedlichen Anforderungen – z. B. bezüglich Schema-Validierung und Versionsrelevanz – werden von diesem Typ Restriktionen gebildet. Solche vom Basisdatentyp Code abgeleitete UML-Klassen werden immer mit dem Stereotyp `xsdCode` (Abschnitt 4.2.4 auf Seite 30) versehen.

Bild 6-2 Verwendung des Stereotyps xsdCode



Es werden vier verschiedene, vom XÖV-Basisdatentyp Code abgeleitete Code-Datentypen unterschieden. Sie stehen für unterschiedliche Arten, eine Codeliste in einen XÖV-Standard einzubinden:

- als *Standard*-Codeliste (Typ 1)
- als *benannte* Codeliste (Typ 2)
- als *versionsfreie* Codeliste (Typ 3)
- als *generische* Codeliste (Typ 4)

Typ 1 und 2 sind so definiert, dass die zu verwendende Codeliste im XÖV-Standard determiniert ist, d. h. die Liste (z. B. Staatentabelle nach ISO 3166) und ihre Version (z. B. Fassung vom 22.04.2007) sind *im Standard* festgelegt, was bei Typ 3 und 4 nicht der Fall ist. Hier wird erst zur Laufzeit durch eine entsprechende Information im Kontext des übermittelten Codes in der Nachrichten-XML-Instanz angegeben, auf welche Version (Typ 3) bzw. auf welche Codeliste und Version (Typ 4), sich der Code bezieht. Typ 1 und 2 unterscheiden sich dadurch, dass in Typ 1 die Einträge der Codeliste explizit im XML-Schema des Standards enthalten sind, während für Typ 2 davon ausgegangen wird, dass die Einträge der Codeliste an anderer Stelle festgelegt sind.

Im Kontext eines XÖV-Standards ergibt sich daraus eine Unterscheidung zwischen internen (Typ 1 und 2) Codelisten und externen (Typen 3 und 4) Codelisten.

In der Konesquenz bedeutet diese enge Bindung **interner** Codelisten an den XÖV-Standard, dass ein neues Release des XÖV-Standards benötigt wird, wenn zur Laufzeit eine aktuellere Codeliste verwendet werden soll. Das ist typischerweise dann der Fall, wenn die Semantik der Codeliste Auswirkungen hat auf die Fachlogik der Verarbeitung in IT-Verfahren, die den XÖV-Standard implementieren.

Ein Beispiel dafür wäre eine Codeliste "Familienstand" ("ledig", "verheiratet", "verwitwet") im Meldewesen: Um die Fachlogik der Verarbeitung von Personendaten zu implementierern, muss bekannt sein, welche Werte der Familienstand einer Person haben kann und welche Konsequenzen ein bestimmter Wert in verschiedenen Kontexten hat. Wäre das der Fall, dann muss in dem entsprechenden XÖV-Standard der Datentyp für die Codeliste "Familienstand" als Typ 1 oder als Typ 2 vom XÖV-Basisdatentyp Code abgeleitet werden.

Typ 1 wird verwendet, wenn die Entscheidung getroffen wurde, dass Nachrichten-Instanzen hinsichtlich des Wertes für Familienstand per Schemavalidierung zu prüfen sind. Auf diese Weise können Nachrichten-Instanzen mit fehlerhaften Familienstandswerten abgewiesen und zurückgesendet werden (return to sender). Typ 2 ist vorzuziehen, wenn die empfangene Nachricht nicht abgewiesen, sondern auf andere Art geprüft bzw. hinsichtlich der übrigen Inhalte ausgewertet werden soll.

Die als **extern** eingebundenen Codelisten sind hingegen vom Release-Zyklus eines XÖV-Standard abgekoppelt. Die Fachlogik des XÖV-Standard ist unabhängig von Varianten der Einträge in einer solchen Liste.

Ein gutes Beispiel hierfür ist eine Staaten-Codeliste. Die Aktualisierungen der Staatenliste kann erfolgen, ohne ein neues Release des XÖV-Standards produzieren zu müssen. In diesem Fall wird die Codeliste "Staat" als Typ 3 oder Typ 4 vom XÖV-Basisdatentype Code abgeleitet und in den XÖV-Standard eingebunden.

In der Regel wird der Typ 3 verwendet, denn somit wird im XÖV-Standard festgelegt, welche Staatenliste (aber nicht, welche Version) zur Laufzeit zu verwenden ist. Alternativ hierzu kann völlig offen gelassen werden, welche konkrete Staatenliste zum Einsatz kommt. Für den Fall z. B., dass der XÖV-Standard einen deutschlandeinheitlichen Rahmen setzt, aber zur Laufzeit z. B. länderspezifische Codelisten zulassen soll. Dann ist Typ 4 relevant.

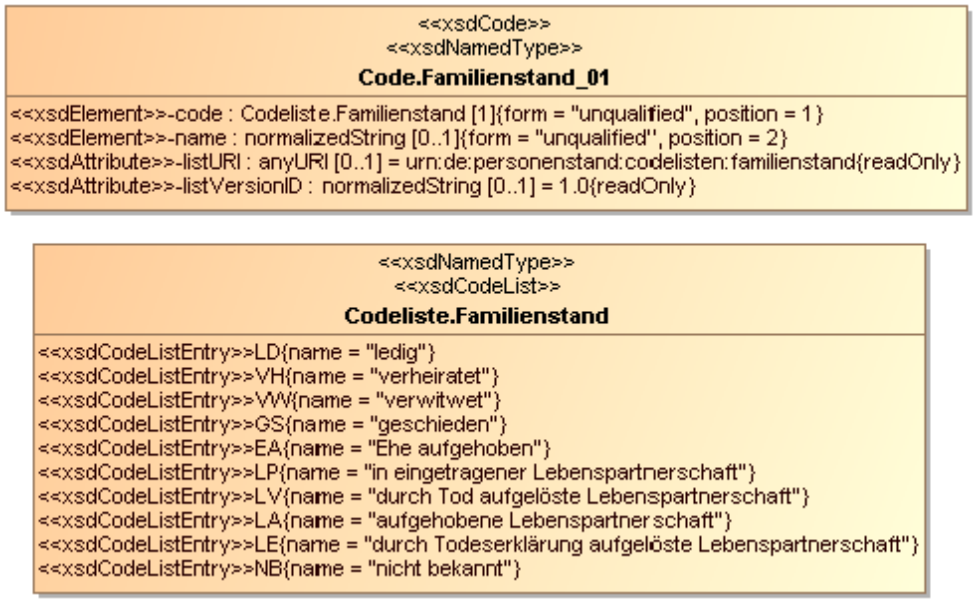
Die Eigenschaften der vier Datentypen Code sind in der nachfolgenden Tabelle im Detail aufgelistet:

	Typ 1	**Typ 2**	**Typ 3**	**Typ 4**
Benennung	Standard-Codeliste	Benannte Codeliste	Versionsfreie Codeliste	Generische Codeliste
Charakter	Codeliste ist Bestandteil des Standards.	Identität und Version der Codeliste sind im Standard determiniert.[1]	Identität der Codeliste ist im Standard determiniert; die jeweils benötigte Version wird hingegen erst zur Laufzeit durch die Applikation in die Nachrichten-Instanz eingetragen.	Weder Identität, noch die Version der Codeliste sind im Standard determiniert; beide werden zur Laufzeit je nach Bedarf angegeben.
Codeliste wird versionsrelevant verwendet?	ja	ja	nein	nein
Codeliste wird schemavalidierend eingesetzt?	ja	nein	nein	nein
code (Datentyp)	konkreter Codelisten-Typ, definiert im Standard (verwendet xs:enumeration)	xs:token	xs:token	xs:token
code (Multiplizität)	1	1	1	1
name (Datentyp)	xs:normalizedString	xs:normalizedString	xs:normalizedString	xs:normalizedString
name (Multiplizität)	0..1	0..1	0..1	0..1
listURI (Datentyp)	xs:anyURI	xs:anyURI	xs:anyURI	xs:anyURI
listURI (Multiplizität)	0..1 (mit fixiertem Wert)	0..1 (mit fixiertem Wert)	0..1 (mit fixiertem Wert)	1 (Wert zur Laufzeit frei wählbar)
listVersionID (Datentyp)	xs:normalizedString (max. 16 Zeichen)	xs:normalizedString (max. 16 Zeichen)	xs:normalizedString (max. 16 Zeichen)	xs:normalizedString (max. 16 Zeichen)
listVersionID (Multiplizität)	0..1 (mit fixiertem Wert)	0..1 (mit fixiertem Wert)	1 (Wert zur Laufzeit frei wählbar)	1 (Wert zur Laufzeit frei wählbar)

1. Für Typ 2 ist zugelassen, die Codeliste mit ihren Codes zu Dokumentationszwecken in das XÖV-UML-Modell des Standards einzutragen (nicht aber in die XML-Schema-Dateien zu übertragen, was Typ 1 entsprechen würde). Das kann sinnvoll sein, wenn die Codeliste zwar nicht schemavalidierend eingesetzt werden soll, sie aber trotzdem mit dem Standard – z. B. als XML-Datei an Verfahrenshersteller – auszuliefern ist.

Nachfolgend sind Beispiele zu den vier Einbindungstypen für Codelisten jeweils mit ihren Konsequenzen für die Darstellung im XÖV-UML-Modell und in den XML-Schema-Dateien angegeben:

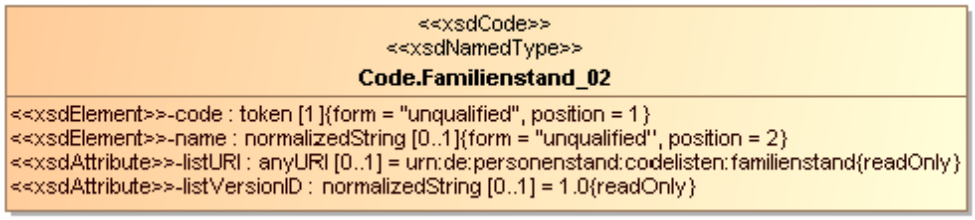
Bild 6-3 Beispiel für Typ 1 – die Standard-Codeliste (UML-Darstellung)



Der entsprechende Typ in XML-Schema (Standard-Codeliste):

```
<xs:complexType name="Code.Familienstand_01">
  <xs:complexContent>
    <xs:restriction base="xoev:Code">
      <xs:sequence>
        <xs:element name="code" form="unqualified">
          <xs:simpleType>
            <xs:restriction base="xs:token">
              <xs:enumeration value="LD"/>
              <xs:enumeration value="VH"/>
              <xs:enumeration value="VW"/>
              <xs:enumeration value="GS"/>
              <xs:enumeration value="EA"/>
              <xs:enumeration value="LP"/>
              <xs:enumeration value="LV"/>
              <xs:enumeration value="LA"/>
              <xs:enumeration value="LE"/>
              <xs:enumeration value="NB"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="name" type="xs:token" form="unqualified" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="listURI" type="xs:anyURI" use="optional"
                                   fixed="urn:de.personenstand.codelisten.familienstand"/>
      <xs:attribute name="listVersionID" type="xs:normalizedString" use="optional"
fixed="1.0"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

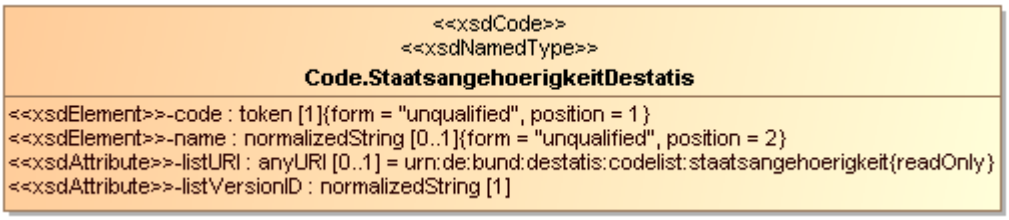
Bild 6-4 Beispiel für Typ 2 – die Benannte Codeliste (UML-Darstellung)



Der entsprechende Typ in XML-Schema (Benannte Codeliste):

```
<xs:complexType name="Code.Familienstand_02">
  <xs:complexContent>
    <xs:restriction base="xoev:Code">
      <xs:sequence>
        <xs:element name="code" type="xs:token" form="unqualified"/>
       <xs:element name="name" type="xs:normalizedString" form="unqualified" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="listURI" type="xs:anyURI" use="optional"
                                   fixed="urn:de.personenstand.codelisten.familienstand"/>
      <xs:attribute name="listVersionID" type="xs:normalizedString" use="optional"
fixed="1.0"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

Bild 6-5 Beispiel für Typ 3 – die Versionsfreie Codeliste (UML-Darstellung)



Der entsprechende Typ in XML-Schema (Versionsfreie Codeliste):

```
<xs:complexType name="Code.StaatsangehoerigkeitDestatis">
  <xs:complexContent>
    <xs:restriction base="xoev:Code">
      <xs:sequence>
        <xs:element name="code" type="xs:token" form="unqualified"/>
       <xs:element name="name" type="xs:normalizedString" form="unqualified" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="listURI" type="xs:anyURI" use="optional"
                                fixed="urn:de.bund.destatis.codelist.staatsangehoerigkeit"/>
      <xs:attribute name="listVersionID" type="xs:normalizedString" use="required"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

Bild 6-6 Beispiel für Typ 4 – die Generische Codeliste (UML-Darstellung)

Der entsprechende Typ in XML-Schema (Generische Codeliste):

```
<xs:complexType name="Code.LaenderspezifischerDokumentTyp">
  <xs:complexContent>
    <xs:restriction base="xoev:Code">
      <xs:sequence>
        <xs:element name="code" type="xs:token" form="unqualified"/>
       <xs:element name="name" type="xs:normalizedString" form="unqualified" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="listURI" type="xs:anyURI" use="required"/>
      <xs:attribute name="listVersionID" type="xs:normalizedString" use="required"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

Je nach Art der Einbindung einer Codeliste sieht der Informationsaustausch in den Nachrichten-XML-Instanzen unterschiedlich aus (in den nachfolgenden Beispielen ist die Darstellung auf die im jeweiligen Fall obligatorischen Elemente beschränkt):

Beispiel: XML-Instanz für Typ 1

```
<xpersonenstand:familienstand_1>
  <code>VH</code>
</xpersonenstand:familienstand_1>


<xpersonenstand:familienstand_1>
  <code>VH</code>
  <name>verheiratet</name>
</xpersonenstand:familienstand_1>


<xpersonenstand:familienstand_1 listURI="urn:de:personenstand:codelisten:familienstand"
listVersionID="1.0">
  <code>VH</code>
</xpersonenstand:familienstand_1>


<xpersonenstand:familienstand_1 listURI="urn:de:personenstand:codelisten:familienstand"
listVersionID="1.0">
  <code>VH</code>
  <name>verheiratet</name>
</xpersonenstand:familienstand_1>
```

URI und Version der Liste müssen in der Nachricht nicht genannt werden, da sie im Standard vollständig determiniert sind.

Beispiel: XML-Instanz für Typ 2

```
<xpersonenstand:familienstand_2>
  <code>VH</code>
</xpersonenstand:familienstand_2>
```

```
<xpersonenstand:familienstand_2>
  <code>VH</code>
  <name>verheiratet</name>
</xpersonenstand:familienstand_2>


<xpersonenstand:familienstand_2 listURI="urn:de:personenstand:codelisten:familienstand"
listVersionID="1.0">
  <code>VH</code>
</xpersonenstand:familienstand_2>


<xpersonenstand:familienstand_2 listURI="urn:de:personenstand:codelisten:familienstand"
listVersionID="1.0">
  <code>VH</code>
  <name>verheiratet</name>
</xpersonenstand:familienstand_2>


<!-- Im Typ 2 werden ungültige Werte nicht im Schema geprüft, das folgende Element ist also
Schema-konform (aber NICHT spezifikationskonform)-->
<xpersonenstand:familienstand_2>
  <code>VERH</code>
</xpersonenstand:familienstand_2>
```

Hier ist auf der Ebene der Nachrichten-Instanz kein Unterschied zu Typ 1 sichtbar. (Der Unterschied besteht lediglich darin, dass bei der Schemavalidierung der Nachricht der Code-Eintrag nicht gegen die Liste der erlaubten Einträge geprüft wird.)

Beispiel: XML-Instanz für Typ 3

```
<xpersonenstand:staatsangehoerigkeit_1 listVersionID="1.0">
  <code>DE</code>
  <name>Deutschland</name>
</xpersonenstand:staatsangehoerigkeit_1>
```

Nur die Version muss in der Nachricht eingetragen sein. Der URI der Liste ist im Standard determiniert.

Beispiel: XML-Instanz für Typ 4

```
<xpersonenstand:laenderspezifischerDokumentTyp_1 listURI="http://xrepository.de/codes/
xdomea-dt-hh" listVersionID="1">
  <code>C12</code>
</xpersonenstand:laenderspezifischerDokumentTyp_1>


<xpersonenstand:laenderspezifischerDokumentTyp_1 listURI="http://xrepository.de/codes/
xdomea-dt-hh" listVersionID="1">
  <code>C12</code>
  <name>Geheimakte</name>
</xpersonenstand:laenderspezifischerDokumentTyp_1>


<!-- Beispiel für die Fallgruppe Typ 4: Liste und Version Teil der Übermittlung, "name" ist
mandatorisch -->
<xpersonenstand:laenderspezifischerDokumentTyp_2 listURI="http://xrepository.de/codes/
xdomea-dt-hh" listVersionID="1">
  <code>C12</code>
  <name>Geheimakte</name>
</xpersonenstand:laenderspezifischerDokumentTyp_2>
```

URI und Version müssen in die Nachricht eingetragen werden, da beide durch den Standard offengelassen werden.

6.2.4 Spezielle Modellierungsmuster

6.2.4.1 Nicht abgeschlossene Codeliste

Die Verwendung einer Codeliste kann mehr oder weniger restriktiv geschehen. Manchmal soll im Zusammenhang eines Code-Datentyps alternativ zu einem Code aus der Codeliste auch die Verwendung eines anderen frei gewählten Wertes gestattet werden, welcher ggf. erst zu einem späteren Zeitpunkt in die Codeliste aufgenommen werden wird bzw. werden kann. Für eine solche Situation steht der Stereotyp `xsdChoice` (siehe Abschnitt 4.2.3 auf Seite 30) zur Verfügung, welcher eine UML-Klasse kennzeichnet. Dem Codelisten-Datentyp kann somit eine Alternative für Freitext zur Seite gestellt werden. In einer Nachrichteninstanz wird dann entweder ein Code aus der Codeliste oder ein anderer, nicht gelisteter Wert übermittelt. Es resultiert also insgesamt ein Datentyp, der die Verwendung einer Codeliste im Rahmen eines nicht abgeschlossenen Moduls zulässt, eine Mischform also aus Code-Datentyp und Freitext.

Bild 6-7 UML-Beispiel für Code-Datentyp zu nicht-abgeschlossener Codeliste

Beispiel: XML-Schema für Code-Datentyp zu nicht-abgeschlossener Codeliste

```
<xs:complexType name="Staatsangehoerigkeit_ListeNichtAbgeschlossen">
  <xs:choice>
    <xs:element name="code" type="xpersonenstand:Code.Staatsangehoerigkeit_Destatis"/>
    <xs:element name="nichtGelisteterWert" type="xs:string"/>
  </xs:choice>
</xs:complexType>
```

Beispiel: XML-Instanz für Code-Datentyp zu nicht-abgeschlossener Codeliste

```
<xpersonenstand:staatsangehoerigkeit_2>
  <xpersonenstand:code listVersionID="1">
    <code>DE</code>
    <name>Deutschland</name>
  </xpersonenstand:code>
</xpersonenstand:staatsangehoerigkeit_2>


<xpersonenstand:staatsangehoerigkeit_2>
  <xpersonenstand:nichtGelisteterWert>Serbien</xpersonenstand:nichtGelisteterWert>
</xpersonenstand:staatsangehoerigkeit_2>
```

6.2.4.2 Unscharfer Codelisten-Eintrag

Für Ergänzungen zu einem übertragenen Code im Sinne einer weiteren Abgrenzung und Präzisierung kann ein ergänzendes Attribut in den Datentyp aufgenommen werden, das für solche Konkretisierungen zur Laufzeit zusätzlich zum übermittelten Code eingesetzt werden kann. Der Code aus der Codeliste ist dann immer anzugeben, die Präzisierung durch den Zusatz ist optional.

Bild 6-8 UML-Beispiel für Code-Datentyp zur Präzisierung unscharfer Codes

Beispiel: XML-Schema zur Präzisierung unscharfer Codes

```
<xs:complexType name="Staatsangehoerigkeit_MitZusatz">
  <xs:sequence>
    <xs:element name="code" type="xpersonenstand:Code.Staatsangehoerigkeit_Destatis"/>
    <xs:element name="zusatz" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

Beipiel: XML-Instanzen zur Präzisierung unscharfer Codes

```
<xpersonenstand:staatsangehoerigkeit_3>
  <xpersonenstand:code listVersionID="1">
    <code>DE</code>
    <name>Deutschland</name>
  </xpersonenstand:code>
</xpersonenstand:staatsangehoerigkeit_3>


<xpersonenstand:staatsangehoerigkeit_3>
  <xpersonenstand:code listVersionID="1">
    <code>DK</code>
    <name>Dänemark</name>
  </xpersonenstand:code>
  <xpersonenstand:zusatz>Grönland</xpersonenstand:zusatz>
</xpersonenstand:staatsangehoerigkeit_3>
```

7. Leitlinien zur Einbindung von XÖV-Kernkomponenten

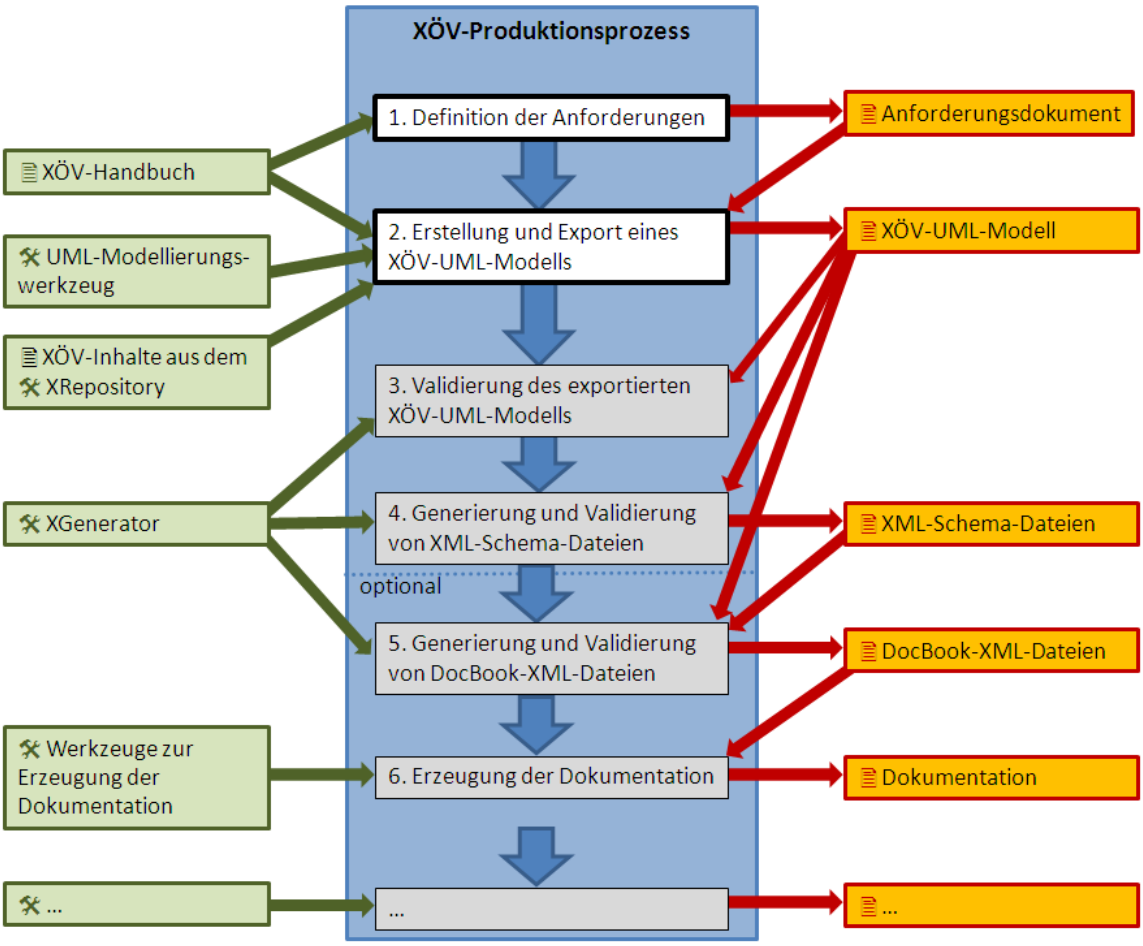
Die durch die XÖV-Koordination im XRepository veröffentlichten Kernkomponenten sollen im XÖV-UML-Modell wiederverwendet werden. Die häufige Wiederverwendung gleicher bzw. ähnlicher Datenstrukturen verbessert die Interoperabilität von XÖV-Standards und erleichtert deren Implementierung.

Die detaillierten Ausführungen zur Wiederverwendung der XÖV-Kernkomponenten in einem XÖV-Standard werden mit der nächsten Version dieses Handbuchs veröffentlicht.

8. Beispielhafte Umsetzung eines XÖV-Standards (XHamsterzucht)

Die erforderlichen Schritte und zu beachtenden Besonderheiten für die Erstellung eines XÖV-Standards sind vielfältig. Das Beispiel befasst sich mit der Modellierung des XÖV-Standards in UML (siehe K-8 auf Seite 17 und K-9 auf Seite 18). Insbesondere wird hierbei aufgezeigt, wie die XÖV-Namens- und Entwurfsregeln (siehe K-10 auf Seite 18 und Abschnitt 5 auf Seite 61) in einem XÖV-UML-Modell umgesetzt werden.

Dieses Kapitel beschreibt die Umsetzung eines XÖV-Standards am **fiktiven** Beispiel einer Hamsterzucht. Das XÖV-UML-Modell zu XHamsterzucht ist im XRepository unter http://www.xrepository.de verfügbar. Zusammen mit den Ergebnissen einer Anforderungsanalyse bildet das UML-Modell damit die ersten beiden Schritte des XÖV-Produktionsprozesses (siehe Abschnitt 3 auf Seite 21) ab. Die Schritte drei bis fünf zur Verarbeitung des UML-Modells sind in Abschnitt 9 auf Seite 131 beschrieben.



8.1 Voraussetzungen

Für ein besseres Verständnis dieses Kapitels sollten folgende Kapitel bereits gelesen worden sein:

- XÖV-Konformitätskriterien (siehe Abschnitt 2 auf Seite 13)
- XÖV-UML-Profil (siehe Abschnitt 4 auf Seite 25)
- XÖV-Namens- und Entwurfsregeln (siehe Abschnitt 5 auf Seite 61)

Das in diesem Kapitel beschriebene UML-Modell steht unter http://www.xoev.de zur Verfügung. Es wurde mit dem UML-Modellierungswerkzeug MagicDraw UML 16.5 erstellt. Zur Anzeige kann der kostenfreie MagicDraw Reader (http://www.magicdraw.com) verwendet werden.

Alle nachfolgenden Abbildungen in diesem Kapitel stammen aus MagicDraw. Die Darstellung in anderen UML-Werkzeugen kann abweichen.

8.2 Definition der Anforderungen

Dieser Abschnitt beschreibt in aller Kürze die Ergebnisse der ersten Phase "Definition der Anforderungen" des XÖV-Produktionsprozesses. Das Anforderungsdokument bildet die Grundlage für die Modellierung des XÖV-Standards XHamsterzucht.

"Unter der Hamsterzucht wird die geplante und durchdachte Vermehrung von Hamstern mit dem Ziel verstanden, Gesundheit, Leistungsvermögen und -bereitschaft sowie bestimmte Rassemerkmale zu erhalten oder zu verbessern.

Jeder in Deutschland lebende Hamster muss gemeldet sein. Jeder Züchter hat die Pflicht, die Geburt eines Hamsters dem Hamsterregister mitzuteilen. Nach der Geburt eines Hamsters erstellen die Züchter daher einen Registrierungsantrag und schicken diesen an das Hamsterregister. Mit dem Registrierungsantrag können zusätzlich alle verwandten Tiere des geborenen Hamsters übermittelt werden. Das Hamsterregister nimmt den Registrierungsantrag entgegen und nimmt die Daten in sein System auf.

Beim Tod eines Hamsters ist neben dem Hamsterregister auch ein Tierarzt und ggf. das Tierseuchenamt zu informieren. Nach dem Tod eines Hamsters erstellen die Züchter daher eine Todesmeldung und senden diese an den Tierarzt. Dieser überprüft den Sterbegrund und entscheidet, ob ein Seuchenverdachtsfall vorliegt. Ist das nicht der Fall, so leitet der Tierarzt die Todesmeldung lediglich an das Hamsterregister weiter. Liegt jedoch ein Seuchenverdachtsfall vor, dann leitet der Tierarzt die Todesmeldung zusätzlich an das Tierseuchenamt weiter. Nach einer nochmaligen dortigen Prüfung löst das Tierseuchenamt ggf. Alarm aus.

Die Hamsterzüchter messen die Erreichung der Hamsterzucht-Ziele, indem sie ihre Tiere in Wettbewerben gegeneinander antreten lassen. An den Wettbewerben dürfen lediglich die Hamster teilnehmen, deren Züchter eine gültige Mitgliedschaft in einem Zuchtverein nachweisen können. Möchte ein Züchter in einem Zuchtverein aufgenommen werden, so erstellt er einen Aufnahmeantrag und schickt diesen an den Zuchtverein. Der Zuchtverein nimmt den Aufnahmeantrag entgegen, prüft diesen und nimmt die neuen Daten in sein System auf. Möchte ein Züchter einen Zuchtverein verlassen, so erstellt er eine Nachricht zur Kündigung der Mitgliedschaft und schickt diese an den Zuchtverein. Der Zuchtverein nimmt die Kündigung entgegen und aktualisiert die Daten in seinem System.

Der Veranstalter eines Wettbewerbs lädt die teilnahmeberechtigen Züchter ein. Nach der Planung einer Veranstaltung erstellt der Veranstalter die Einladungen und verschickt sie an die Züchter. Die Züchter registrieren den Eingang der Einladung. Möchte ein Züchter mit einem Hamster an einem Wettbewerb

teilnehmen, so erstellt er einen Teilnahmeantrag und sendet ihn an den Veranstalter des Wettbewerbs. Dieser überprüft den Antrag auf seine Richtigkeit und fügt die neuen Daten seinem System hinzu. Nach einem Wettbewerb wird den ursprünglich eingeladenen Teilnehmern ihr Ergebnis mitgeteilt. Hierzu verschickt der Veranstalter eine Nachricht mit den Ergebnissen der Teilnehmer. Fällt der Wettbewerb aus, dann ist der Grund für den Ausfall des Wettbewerbs anstatt der Wettbewerbsergebnisse zu übermitteln. Die Züchter registrieren die veröffentlichen Ergebnisse und fügen die neuen Daten ihrem System hinzu."
(Auszug aus dem fiktiven Anforderungsdokument)

8.3 Erstellung des XÖV-UML-Modells

In den XÖV-Konformitätskriterien (siehe Abschnitt 2 auf Seite 13) ist die Modellierung der Prozesse (K-8) und Datenstrukturen (K-9) in UML benannt. Dieser Abschnitt beschreibt die Modellierung der aus den zuvor genannten Anforderungen resultierenden Prozess- und Datenstrukturen vor dem Hintergrund der zu berücksichtigenden Namens- und Entwurfsregeln (siehe Abschnitt 5 auf Seite 61).

8.3.1 Allgemeine Regeln

Bei der Erstellung des UML-Modells sind folgende allgemeine Namens- und Entwurfsregeln zu beachten:

- Erlaubte Zeichen für Namen (siehe NDR-11 auf Seite 68)
- Erlaubte Zeichen für Klassifikationen in Namen (siehe NDR-12 auf Seite 69)
- Namen in deutscher Sprache (siehe NDR-14 auf Seite 69)
- Groß- und Kleinschreibung von (und in zusammengesetzten) Namen (siehe NDR-15 auf Seite 70)
- Dokumentation in deutscher Sprache (siehe NDR-19 auf Seite 72)

8.3.2 Aufbau der UML-Modell-Struktur

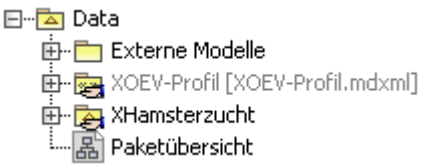
Beim Aufbau der UML-Modell-Struktur sind folgende Namens- und Entwurfsregeln zu beachten:

- Hauptstruktur des UML-Modells (siehe NDR-2 auf Seite 63)
- Detaillierte Struktur des UML-Modells (siehe NDR-5 auf Seite 65)

Die folgenden Abschnitte gehen detailliert auf die Umsetzung der benannten Regeln ein.

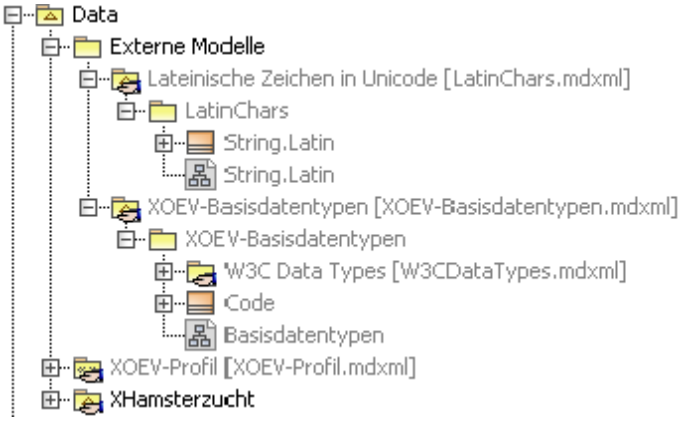
8.3.2.1 Hauptstruktur

Die Hauptstruktur von XHamsterzucht ist wie folgt abgebildet:



Das UML-Modell enthält das Modell "XHamsterzucht" und das Paket "Externe Modelle". Gemäß den XÖV-Vorgaben ist das Modell "XHamsterzucht" nach dem XÖV-Standard benannt und besitzt den Stereotyp `xsdXModel` (siehe nachfolgende Abbildung).

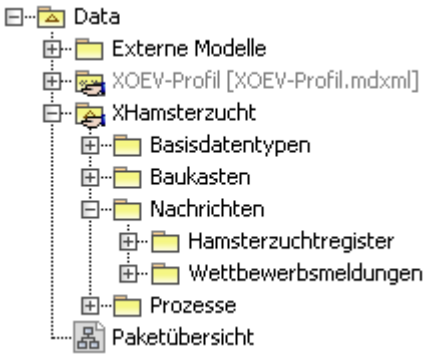
Unterhalb des Pakets "Externe Modelle" befinden sich das Modell "Lateinische Zeichen in Unicode", welches den Datentyp "String.Latin" enthält, sowie das Modell "XOEV-Basisdatentypen" (siehe Abschnitt 4.3 auf Seite 54), das seinerseits das Paket "W3C Data Types" beinhaltet.



Diese Strukturen bilden die Basis für die Modellierung der Inhalte unterhalb des Modells "XHamsterzucht" sowie deren spätere Verarbeitung durch das XÖV-Produktionszubehör.

8.3.2.2 Detaillierte Struktur

Unterhalb des Pakets "XHamsterzucht" befindet sich der projektspezifische Bestandteil des gleichnamigen XÖV-Standards.



Hierzu gehören die Pakete "Basisdatentypen", "Baukasten", "Nachrichten" und "Prozesse".

Unterhalb des Pakets "Nachrichten" befinden sich zwei weitere Pakete: "Hamsterzuchtregister" und "Wettbewerbsmeldungen". Diese Pakete stellen die zwei Hauptgruppen von XHamsterzucht dar.

Alle zuvor genannten Pakete (bis auf "Prozesse" und "Nachrichten") besitzen den Stereotyp `xsd-Schema`.

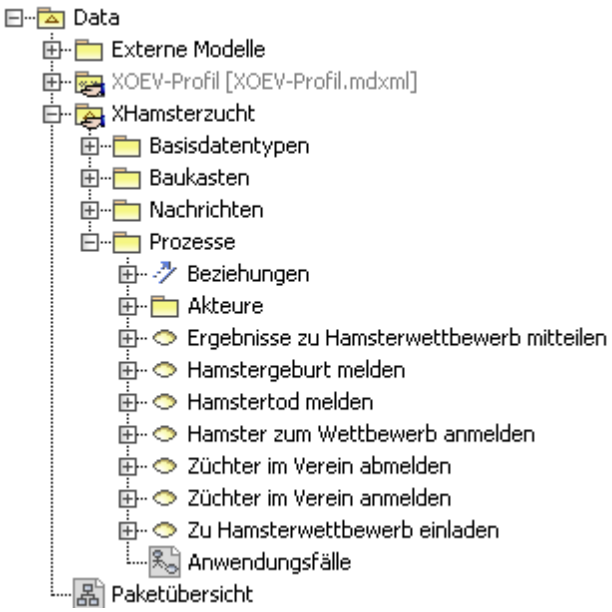
Das Paket "Prozesse" dient lediglich zu Dokumentationszwecken. Es enthält Anwendungsfälle und Aktivitätsdiagramme zu den Nachrichten, die im Rahmen des Standards ausgetauscht werden.

8.3.3 Abbildung der Anwendungsfälle

Gemäß den XÖV-Konformitätskriterien (siehe Abschnitt 5 auf Seite 61) sind die in diesem Kapitel genannten Anforderungen (siehe Abschnitt 8.2 auf Seite 104) als Anwendungsfälle in UML-Aktivitätsdiagrammen abgebildet.

Hinweis: Grundsätzlich bleibt es dem Modellierer überlassen, ob die Anwendungsfälle des XÖV-Standards in einem (zentral) oder in mehreren Paketen (verteilt) abgebildet werden. Wenn die Prozesse eines XÖV-Standards sehr komplex sind, empfiehlt sich eine Aufteilung in mehrere Pakete (beispielsweise in die jeweiligen Hauptgruppen).

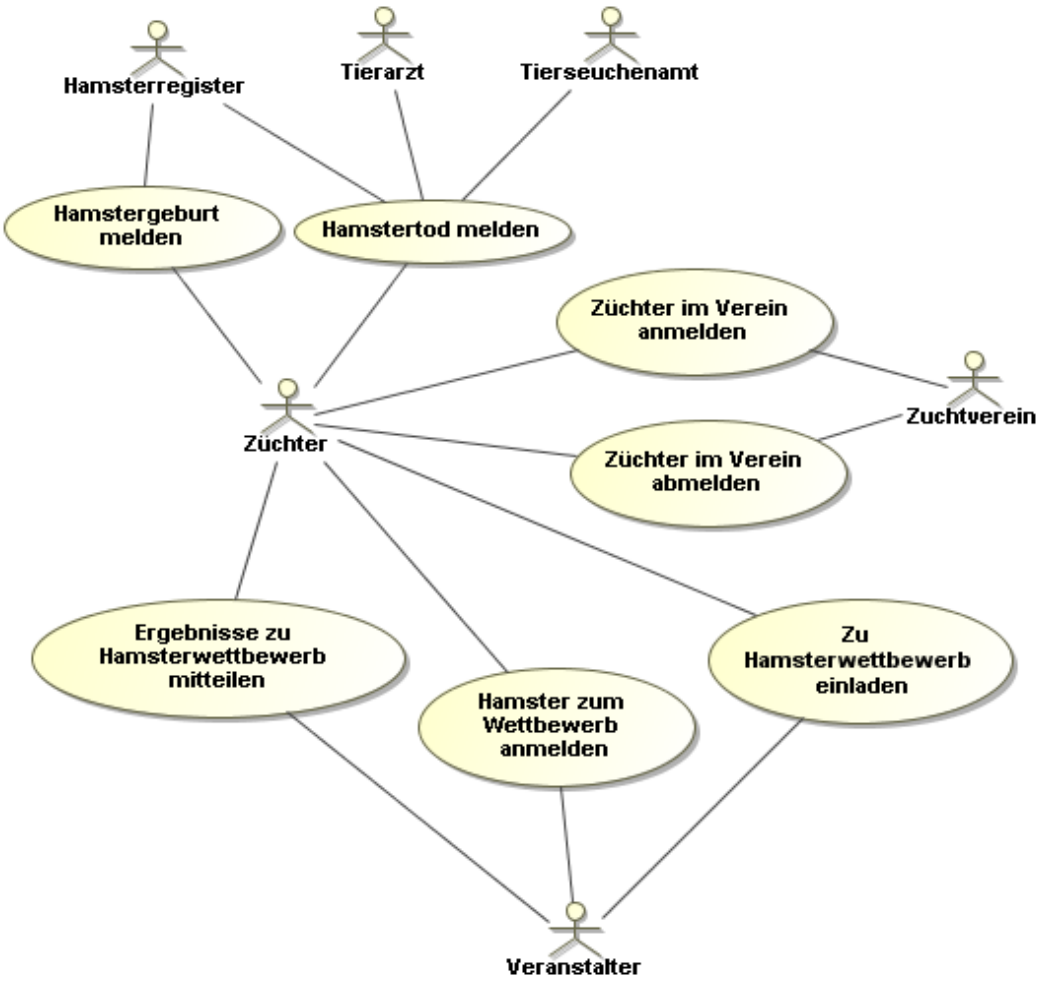
Für die Modellierung von XHamsterzucht wurde aufgrund der geringen Komplexität die zentrale Ablage aller Anwendungsfälle im Paket "Prozesse" des Modells "XHamsterzucht" gewählt.



Anhand der Anforderungen konnten die folgenden Akteure identifiziert werden:

- Hamsterregister
- Tierarzt
- Tierseuchenamt
- Veranstalter
- Züchter
- Züchterverein

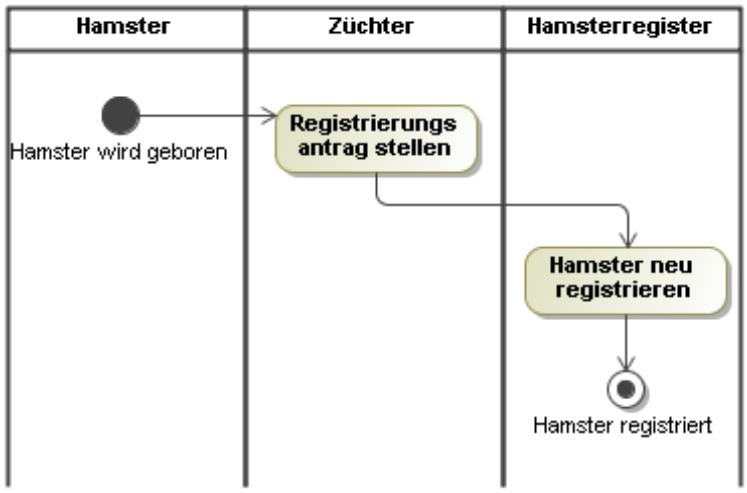
Diese Akteure sind in verschiedene Anwendungsfälle involviert:



Die folgenden Abschnitte beschreiben die Aktivitätsdiagramme zu den einzelnen Anwendungsfällen.

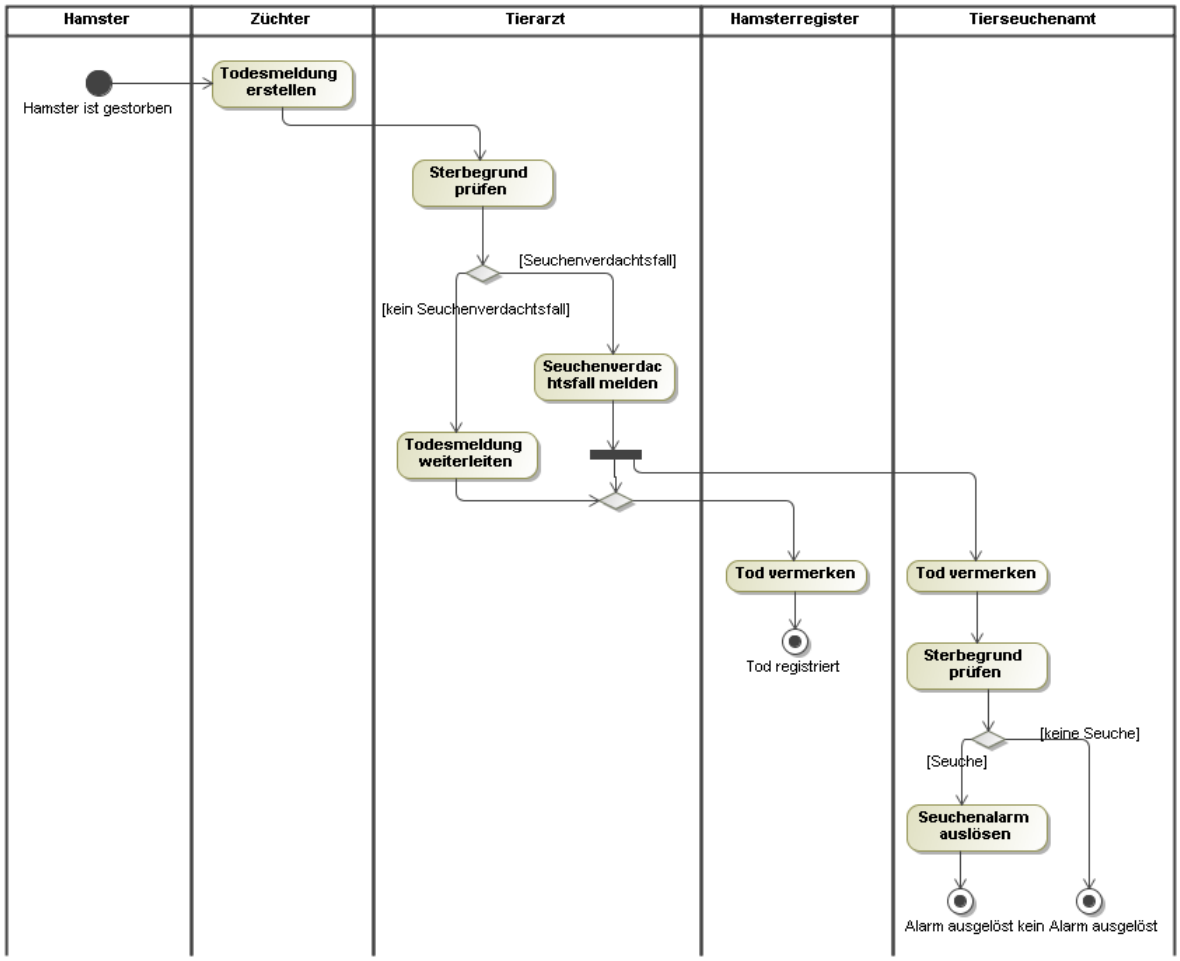
8.3.3.1 Anwendungsfall "Hamstergeburt melden"

Ein Züchter meldet die Geburt eines Hamsters an das Hamsterregister:



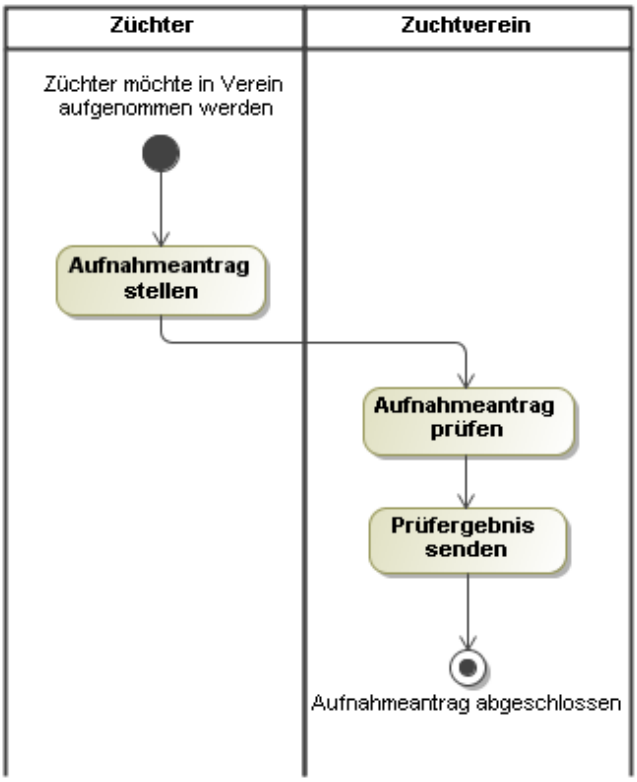
8.3.3.2 Anwendungsfall "Hamstertod melden"

Im Falle des Todes eines Hamsters meldet der Züchter den Sachverhalt an einen Tierarzt:



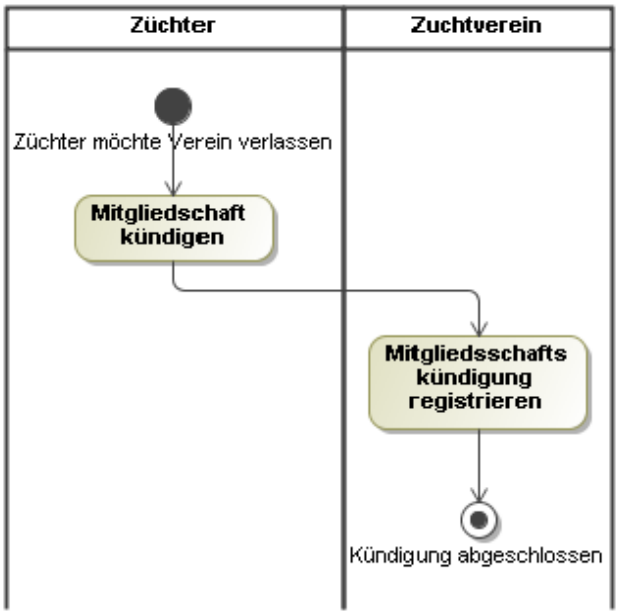
8.3.3.3 Anwendungsfall "Züchter im Verein anmelden"

Ein Züchter meldet sich in einem Züchterverein an:



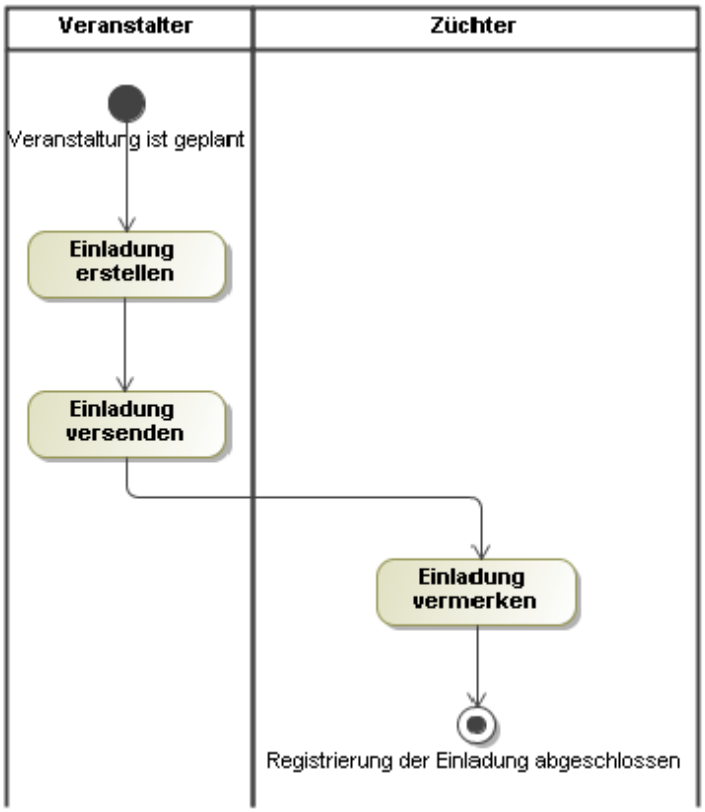
8.3.3.4 Anwendungsfall "Züchter im Verein abmelden"

Ein Züchter meldet sich in einem Züchterverein ab:



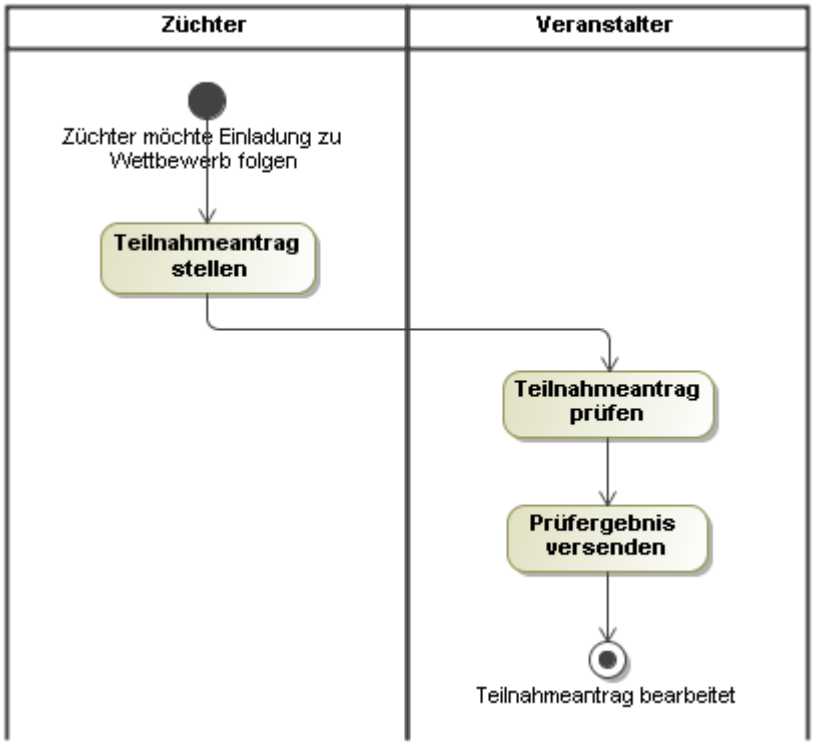
8.3.3.5 Anwendungsfall "Zu Hamsterwettbewerb einladen"

Ein Veranstalter lädt einen Züchter zu einem Wettbewerb ein:



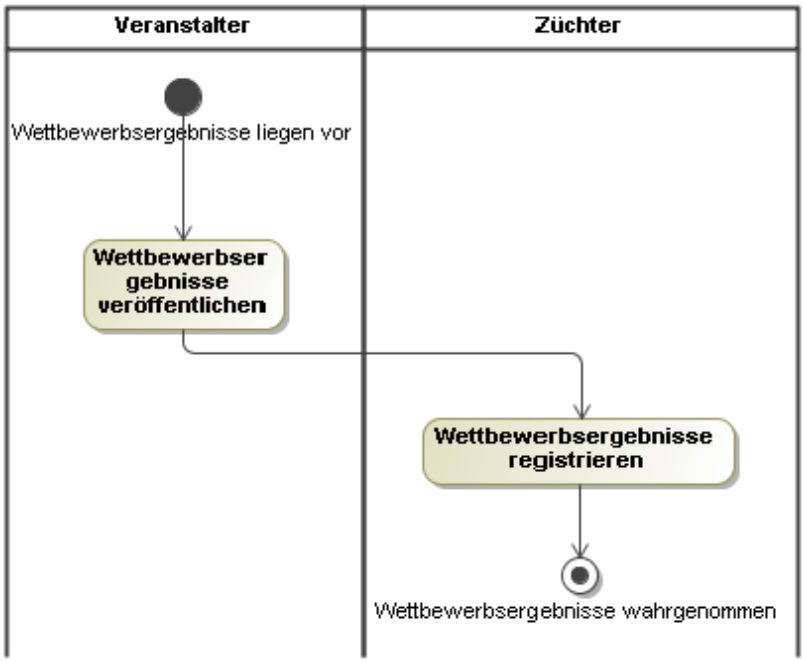
8.3.3.6 Anwendungsfall "Hamster zum Wettbewerb anmelden"

Der eingeladene Züchter meldet seinen Hamster zum Wettbewerb an:



8.3.3.7 Anwendungsfall "Ergebnisse zu Hamsterwettbewerb mitteilen"

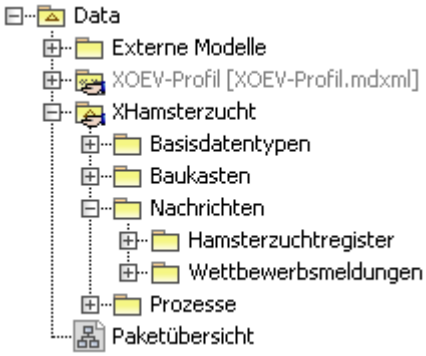
Der Veranstalter schickt dem teilnehmenden Züchter nach Beendigung des Wettbewerbs das Ergebnis zu:



8.3.4 Abbildung von globalen Elementen

Die in den vorangegangenen Abschnitten beschriebenen Anwendungsfälle beinhalten den Austausch von elektronischen Nachrichten in Form von XML-Dokumenten.

Die Nachrichten lassen sich den beiden Hauptgruppen "Hamsterzuchtregister" und "Wettbewerbsmeldungen" zuordnen. Diese Hauptgruppen finden sich in der Paketstruktur entsprechend wieder:



Unterhalb der Hauptgruppen werden die Nachrichten als UML-Klassen angelegt. Bei der Modellierung dieser UML-Klassen sind die folgenden Regeln zu beachten:

- Nachrichten als globale Elemente (siehe NDR-3 auf Seite 64)
- Eindeutige versionsübergreifende Namen von Nachrichten (siehe NDR-13 auf Seite 69)

- Namensstruktur von globalen Elementen (siehe NDR-16 auf Seite 70)
- Eindeutige versionsübergreifende Nummern in Namen von Nachrichten (siehe NDR-17 auf Seite 71)
- Dokumentation der Rechtsgrundlagen (siehe NDR-20 auf Seite 72)

Die folgenden Abschnitte gehen auf die Umsetzung der benannten Regeln ein.

8.3.4.1 Namensgebung

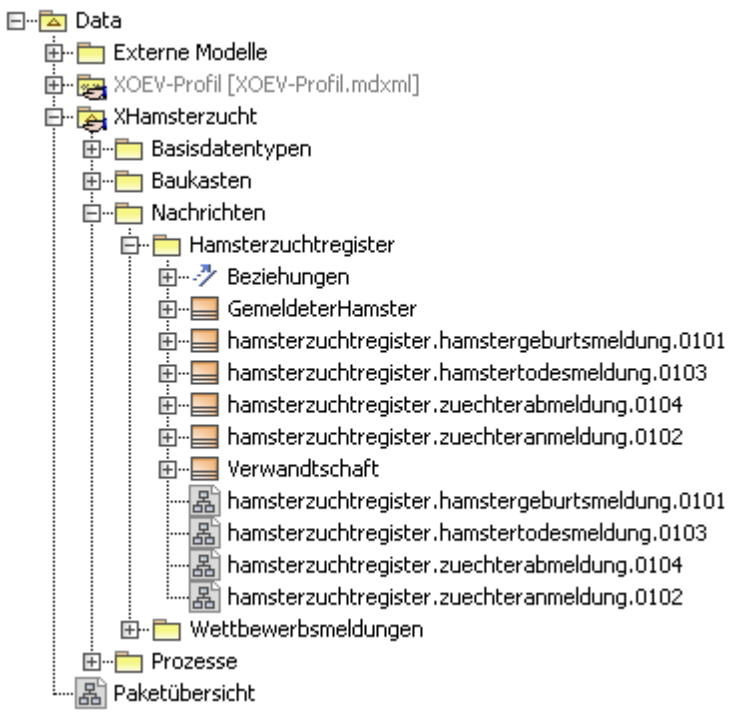
Der Name der UML-Klasse einer Nachricht sollte nach dem folgendem Muster aufgebaut sein:

[hauptgruppe].[nachrichtenname].[nachrichtennummer]

Für die UML-Klasse einer Nachricht des Anwendungsfalls "Züchter im Verein anmelden" ergibt sich daher bspw. folgender Name:

hamsterzuchtregister.zuechteranmeldung.0102

Nach diesem Muster sind alle übrigen Nachrichten unterhalb der Pakete für die Hauptgruppen "Hamsterzuchtregister" und "Wettbewerbsmeldungen" aufgebaut:



8.3.4.2 Modellierung

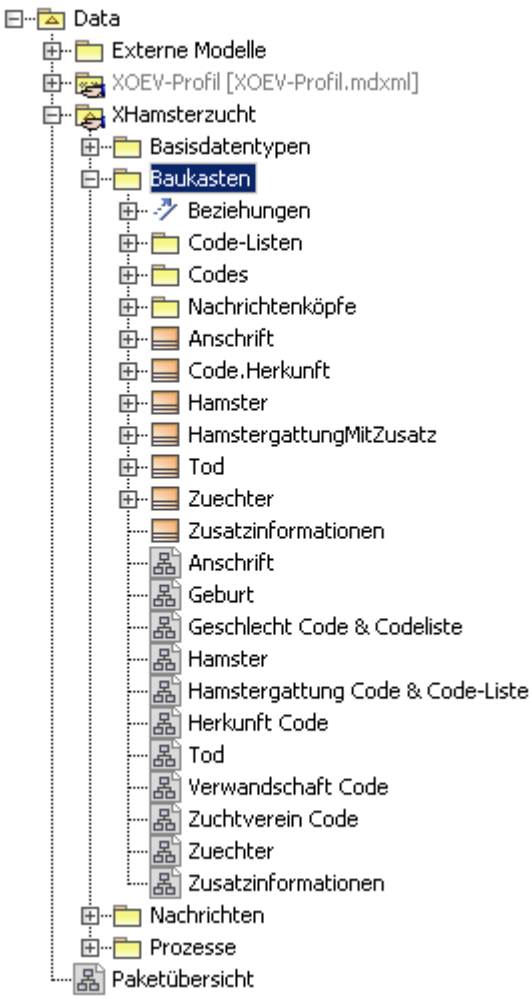
Die UML-Klasse der Nachricht "hamsterzuchtregister.zuechteranmeldung.0102" ist mit dem Stereotyp `xsdGlobalElement` ausgezeichnet. Dadurch ist sie mit den nach der Verarbeitung durch das XÖV-Produktionszubehör entstandenen XML-Schemata über ein XML-Schema-Root-Element als XML-Dokument instanziierbar.

Zusätzlich erhält sie zu Dokumentationszwecken den Stereotyp `xsdMessage`. Eine der dadurch zuweisbaren Stereotyp-Eigenschaften ist z.B. die Rechtsgrundlage. Für die Nachricht "hamsterzuchtregister.zuechteranmeldung.0102" wurde die "Züchterverordnung" als Rechtsgrundlage dokumentiert.

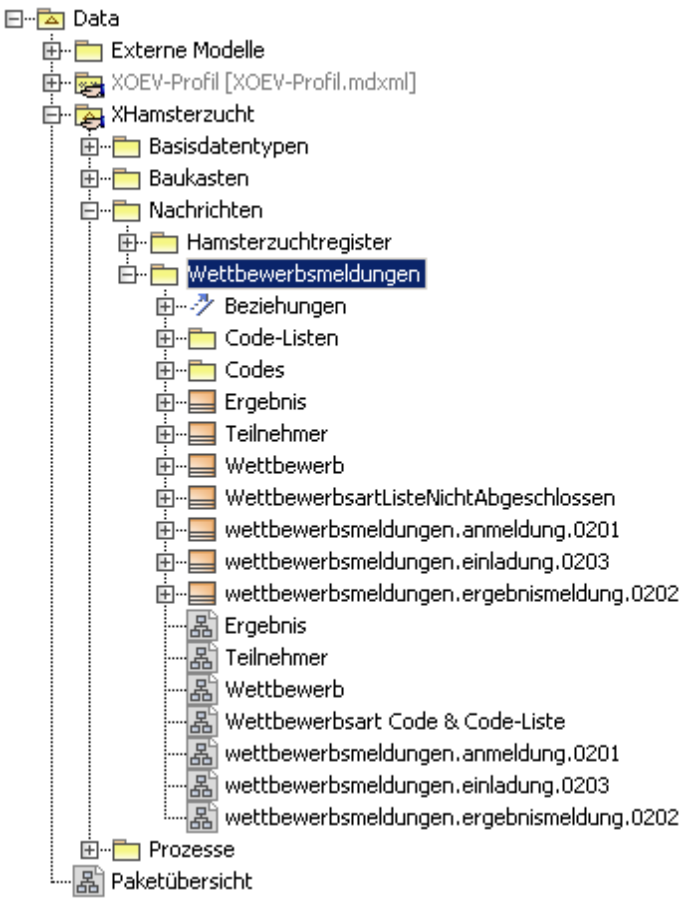
8.3.5 Abbildung der fachlichen Datentypen

Fachliche Datentypen bestimmen die komplexen Strukturen der Nachrichteninhalte. Sie sind aus mehreren einfachen Typen und ggf. weiteren fachlichen Datentypen zusammengesetzt.

Ein Großteil der in XHamsterzucht erstellten fachlichen Datentypen befindet sich unterhalb des Pakets "Baukasten", da sie von Nachrichten aus mehreren Hauptgruppen verwendet werden:



In XHamsterzucht existieren jedoch auch fachliche Datentypen, die nur innerhalb einer bestimmten Hauptgruppe genutzt werden. Diese spezifischen Datentypen wurden im Paket "Nachrichten" unterhalb des jeweiligen Hauptgruppen-Pakets, z.B. "Wettbewerbsmeldungen", abgelegt:



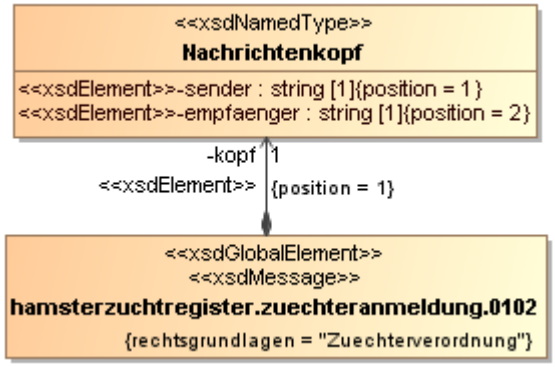
Bei der Erstellung der UML-Klassen für fachliche Datentypen sind folgende Regeln zu beachten:

- Nutzung von XML-Attributen und XML-Elementen (siehe NDR-6 auf Seite 66)
- XML-Wildcard-Elemente mit Namensraum (siehe NDR-7 auf Seite 67)
- Umgang mit Restriktionen über unterschiedliche Namensräume (siehe NDR-23 auf Seite 73)
- Wiederverwendung generischer Nachrichten-Eigenschaften (siehe NDR-24 auf Seite 74)
- Abgrenzung der Wiederverwendung durch Komposition und Restriktion (siehe NDR-25 auf Seite 74)

Die folgenden Abschnitte liefern eine Beschreibung zur Umsetzung der benannten Regeln.

8.3.5.1 Wiederverwendung von Datentypen

Häufig wiederkehrende Bestandteile sollten als wiederverwendbare UML-Klassen modelliert werden. Ein gutes Beispiel hierfür ist der Nachrichtenkopf. Er ist vergleichbar mit einem Briefkopf bestehend aus Absender und Empfänger, der jeder Nachricht hinzugefügt wird.

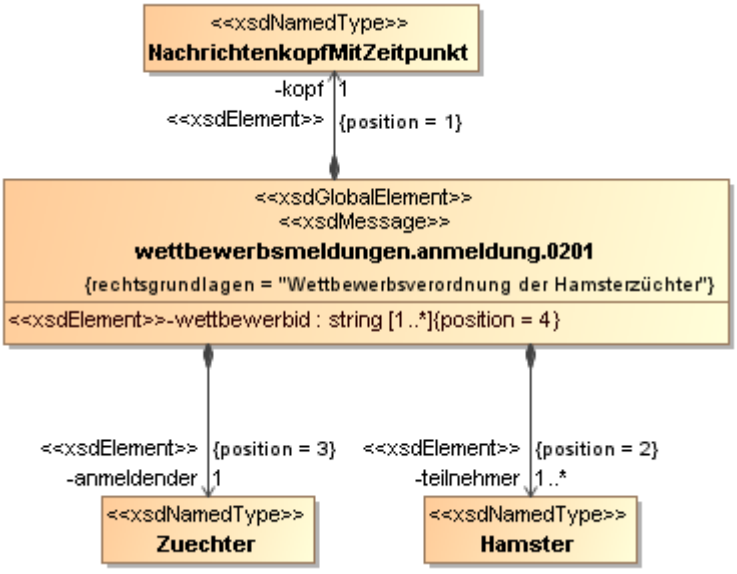


Die entsprechende UML-Klasse, die den Nachrichtenkopf repräsentiert, wird mit dem Stereotyp `xsdNamedType` versehen. Dadurch wird sie auf XML-Schema-Ebene als benannter Datentyp umgesetzt. Auf diese Weise kann jede UML-Klasse für eine Nachricht durch eine Assoziation mit der UML-Klasse des Nachrichtenkopfes verbunden werden.

Die Abbildung zeigt das am Beispiel der Nachricht "hamsterzuchtregister.zuechteranmeldung.0102". Die Assoziation wurde mit dem Stereotyp `xsdElement` versehen, so dass daraus im XML-Schema ein XML-Element mit einer Referenz auf den komplexen Typ "Nachrichtenkopf" erzeugt wird.

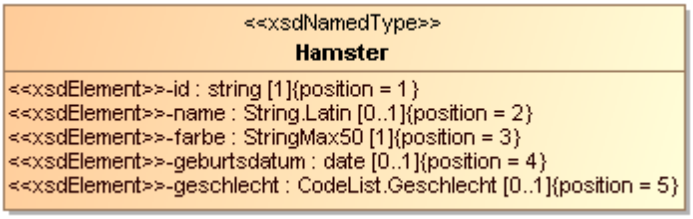
8.3.5.2 Komposition zur Bildung komplexer Datenstrukturen

Komplexe Strukturen sollten aus bereits bestehenden Komponenten zusammengesetzt werden. Die Nachricht "wettbewerbsmeldungen.anmeldung.0201" wird aus den wiederverwendbaren UML-Klassen "NachrichtenkopfMitZeitpunkt", "Zuechter" und "Hamster" zusammengesetzt:



8.3.5.3 Nutzung von XML-Elementen

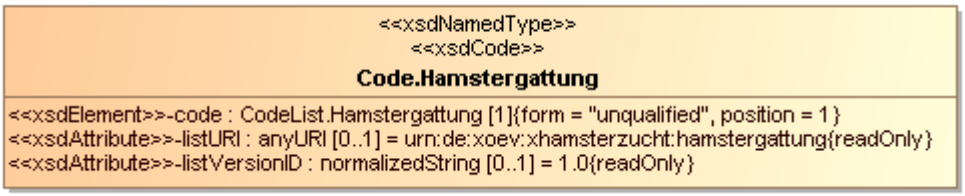
Da es sich bei allen Eigenschaften des Hamsters um fachliche Informationen handelt, ist ihrer Repräsentation in der entsprechenden UML-Klasse der Stereotyp `xsdElement` zugewiesen. Im XML-Schema werden daraus XML-Elemente erzeugt.



Weiterhin wird den Werten in der Eigenschaft "position" des Stereotyps `xsdElement` eine eindeutige und fortlaufende Nummer zugewiesen. Diese Nummer wird für die Festlegung der Reihenfolge der XML-Elemente innerhalb eines Datentyps bei der Generierung der XML-Schemata durch das XÖV-Produktionszubehör benötigt.

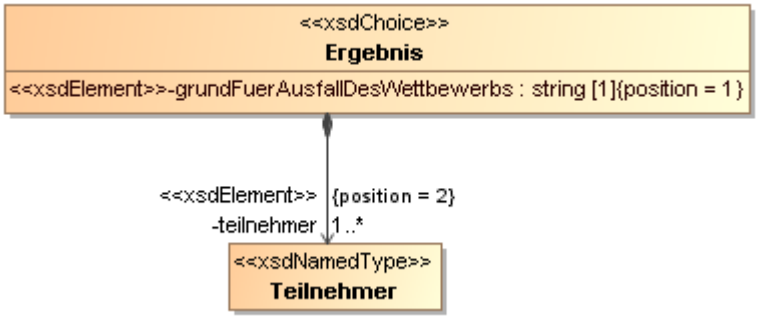
8.3.5.4 Nutzung von XML-Attributen

Da es sich bei den Eigenschaften "listURI" und "listVersionID" des Datentyps "Code.Hamstergattung" um technische Informationen handelt, ist ihrer Repräsentation in der entsprechenden UML-Klasse der Stereotyp `xsdAttribute` zugewiesen. Im XML-Schema werden daraus XML-Attribute erzeugt.



8.3.5.5 Auswahl sich ausschließender XML-Elemente

Bei der Übermittlung der Wettbewerbsergebnisse ist wahlweise der Grund für den Ausfall des Wettbewerbs oder die Teilnehmer mit ihrer Platzierung zu übermitteln.



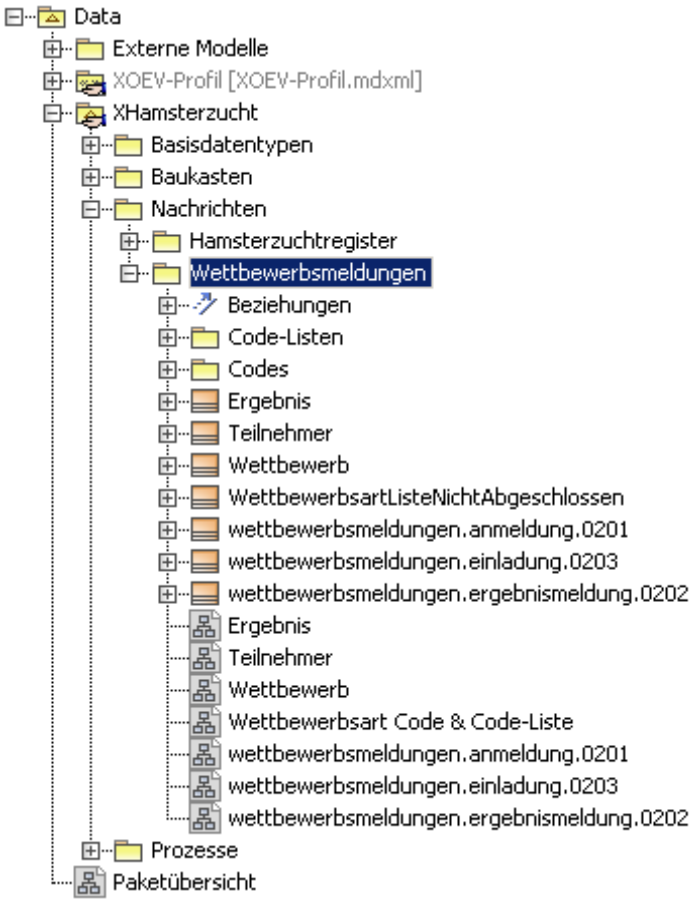
In XHamsterzucht wird daher die UML-Klasse "Ergebnis" mit dem Stereotyp `xsdChoice` versehen. Dadurch schließt sich eine gemeinsame Verwendung der mit dem Stereotyp `xsdElement` versehenen Eigenschaften aus: entweder kann in einer XML-Nachricht das XML-Element "grundFuerAusfallDes-Wettbewerbs" oder mehrere XML-Elemente des Typs "Teilnehmer" übermittelt werden.

Hinweis: In dem obigen Beispiel gibt es keine mit dem Stereotyp `xsdAttribute` versehenen Eigenschaften. Ein Modellierer kann diese `xsdAttribute`-Eigenschaften einer mit dem Stereotyp `xsdChoice` versehenen UML-Klasse zuweisen. Diese werden dann jedoch in jedem Fall in der XML-Nachricht angegeben und nicht bei der Auswahl sich ausschließender XML-Elemente berücksichtigt.

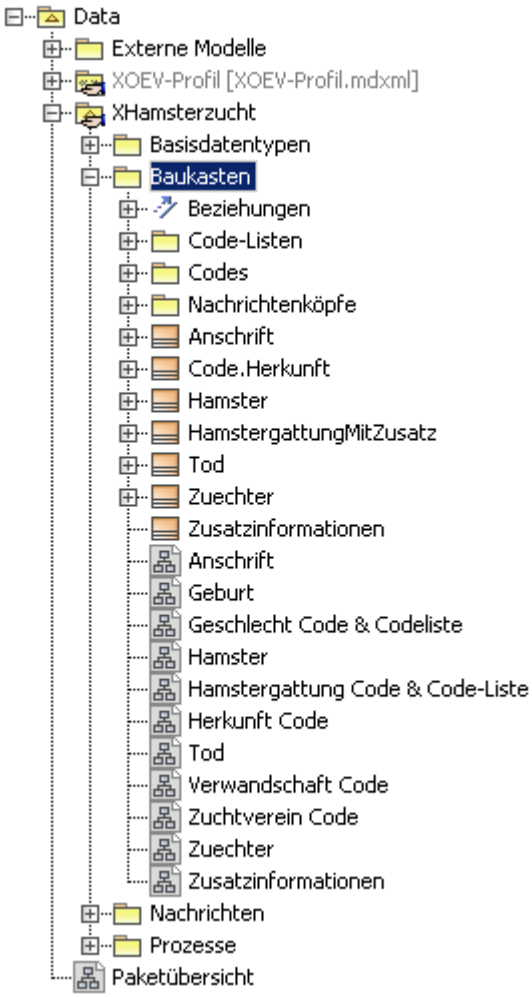
8.3.6 Abbildung der Codelisten

Codelisten dienen der semantischen Vereinheitlichung zentraler fachlicher Begriffe und stellen im XÖV-Kontext spezielle fachliche Datentypen dar. Sie bestehen im UML-Modell aus einer UML-Enumeration und/oder einer UML-Klasse (mindestens jedoch einer UML-Klasse).

Mit der Einladung zu einem Wettbewerb übermittelt der Veranstalter neben den üblichen Daten zum Wettbewerb auch die Wettbewerbsart. Die Eigenschaft Wettbewerbsart kann lediglich Werte einer konkreten Liste von Codes annehmen – wobei nicht ausgeschlossen ist, dass in der Zukunft der Liste noch weitere Werte hinzugefügt werden. In XHamsterzucht sind das die Werte "1" und "2". Diese endliche Liste wird im UML- Modell eines XÖV-Standards als UML-Enumeration mit dem Sterotypen `xsdCodeList` modelliert. Die Codeliste für die Wettbewerbsart befindet sich im Paket "Code-Listen" unterhalb des Pakets der Hauptgruppe "Wettbewerbsmeldungen". Die zu dieser Codeliste gehörende UML-Klasse mit der Beschreibung der Metadaten, z.B. Version und URL, wird mit dem Stereotyp `xsdCode` gekennzeichnet und befindet sich im Paket "Codes" unterhalb des Pakets "Wettbewerbsmeldungen".



Die bislang betrachtete Codeliste "CodeList.Wettbewerbsart" ist eine Codeliste, die lediglich im Kontext der Wettbewerbsmeldungen benötigt wird. Daher wurden ihre Bestandteile unterhalb des Pakets "Wettbewerbsmeldungen" angelegt. In mehreren Hauptgruppen genutzte Codelisten befinden sich in den Paketen "Code-Listen" und "Codes" unterhalb des Pakets "Baukasten".



Bei der Modellierung der UML-Bestandteile von Codelisten sind folgende Regeln zu beachten:

- Erlaubte Einbindungsarten für Codelisten (siehe NDR-4 auf Seite 64)
- Eindeutige versionsübergreifende Codes in Codelisten (siehe NDR-8 auf Seite 67)
- Umgang mit unscharfen Codelisten-Einträgen und nicht abgeschlossenen Codelisten (siehe NDR-9 auf Seite 67)
- Codenamen für Codelisten-Einträge (siehe NDR-21 auf Seite 73)
- Unveränderte Übernahme von XÖV-Codelisten (siehe NDR-22 auf Seite 73)

Die folgenden Abschnitte gehen auf die Umsetzung der benannten Regeln ein.

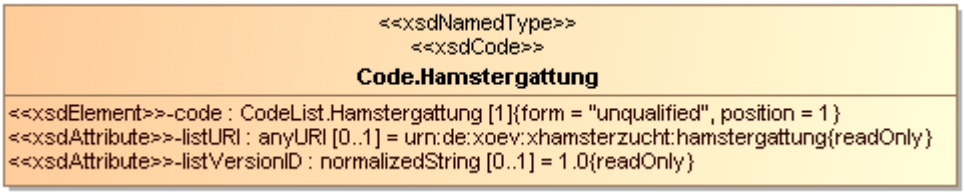
8.3.6.1 Einbindungsarten für Codelisten

Aus den "Leitlinien zu Codelisten" (siehe Abschnitt 6 auf Seite 79) geht hervor, dass für verschiedene Anwendungskontexte entsprechende Einbindungsarten für Codelisten existieren.

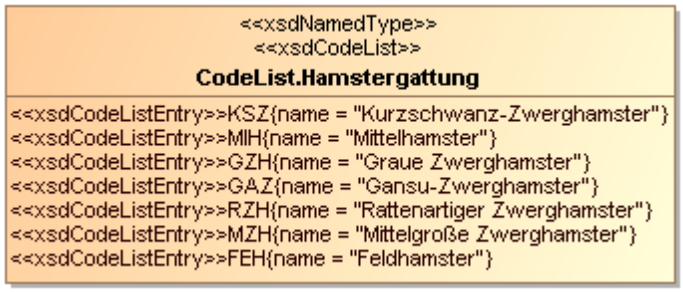
Je nach Einbindungsart muss für eine Codeliste im UML-Modell eine mit dem Stereotyp `xsdCode` versehene UML-Klasse und ggf. eine mit dem Stereotyp `xsdCodeList` versehende Enumeration angelegt werden. Analog zu den UML-Klassen der fachlichen Datentypen wurden die UML-Klassen der Code-Datentypen in XHamsterzucht zusätzlich mit dem Stereotyp `xsdNamedType` versehen. Dadurch werden sie im XML-Schema benannt und können an verschiedenen Stellen wiederverwendet werden.

Im Folgenden wird anhand der Code-Datentypen aus XHamsterzucht die Modellierung der vier möglichen Einbindungsarten für Codelisten vorgestellt.

8.3.6.1.1 Code-Datentyp "Code.Hamstergattung" (Typ 1)

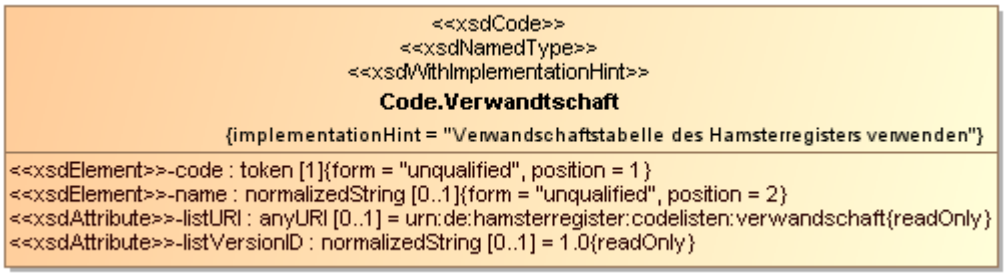


An den mit readOnly[1] versehenen Eigenschaften "listURI" und "listVersionID" der abgebildeten UML-Klasse ist zu erkennen, dass es sich bei diesem Code-Datentyp nicht um Typ 3 (versionsfeie Codeliste) und nicht um Typ 4 (generische Codeliste) handelt. Der Eigenschaft "code" ist als Datentyp die Enumeration "CodeList.Hamstergattung" zugewiesen - somit liegt auch nicht Typ 2 (benannte Codeliste) vor.



Da die UML-Enumeration der Codeliste Bestandteil des UML-Modells des Standards ist, handelt es sich hier um eine Standard-Codeliste (Typ 1).

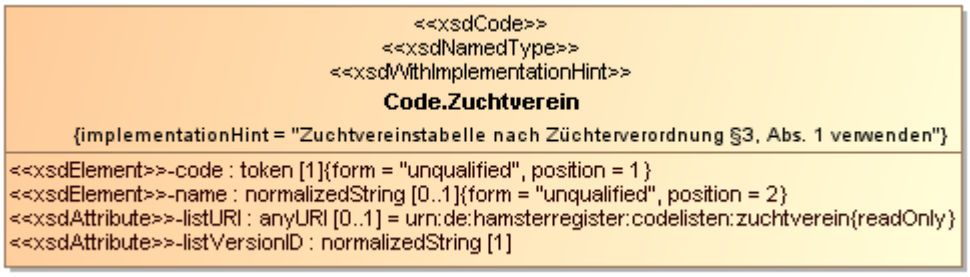
8.3.6.1.2 Code-Datentyp "Code.Verwandtschaft" (Typ 2)



1. festgelegte (fixed) Werte im XML-Schema, die in der XML-Instanz nicht abweichen dürfen

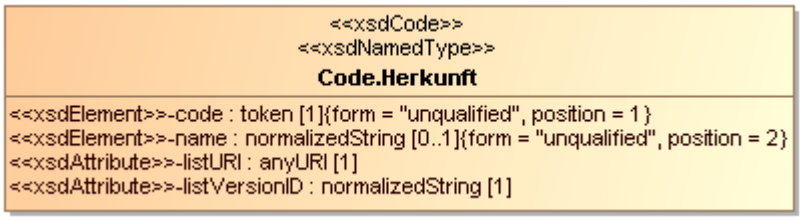
Bei diesem Code-Datentyp sind die Eigenschaften "listURI" und "listVersionID" der abgebildeten UML-Klasse ebenfalls vorbelegt und mit "readOnly" versehen. Im Gegensatz zum Code-Datentyp "Code.Hamstergattung" (Typ 1) ist für die Eigenschaft "code" kein Datentyp einer Codeliste angegeben. Stattdessen verweist dieser Typ auf den XÖV-Basisdatentyp "token". Es handelt sich daher um Typ 2 (benannte Codeliste).

8.3.6.1.3 Code-Datentyp "Code.Zuchtverein" (Typ 3)



Der Code-Datentyp "Code.Zuchtverein" ist lediglich durch die Eigenschaften "listURI" der abgebildeten UML-Klasse genauer spezifiziert. Es handelt sich daher um Typ 3 (versionsfreie Codeliste).

8.3.6.1.4 Code-Datentyp "Code.Herkunft" (Typ 4)



Beim Code-Datentyp "Code.Herkunft" sind weder die Eigenschaften "listURI" und "listVersionID" noch die Eigenschaft "code" der abgebildeten UML-Klasse vorbelegt. Es handelt sich daher um Typ 4 (generische Codeliste).

8.3.6.2 Modellierung einer Codeliste

Im UML-Modell werden die Codelisten von XHamsterzucht durch UML-Enumerationen repräsentiert. Sie beinhalten alle Einträge der jeweiligen Codeliste als UML-Enumerations-Literale und sind für den Gebrauch in einem Code-Datentyp vom Typ 1 innerhalb des Standards gedacht. Zur Kennzeichnung einer Codeliste wird der Stereotyp `xsdCodeList` zugewiesen.

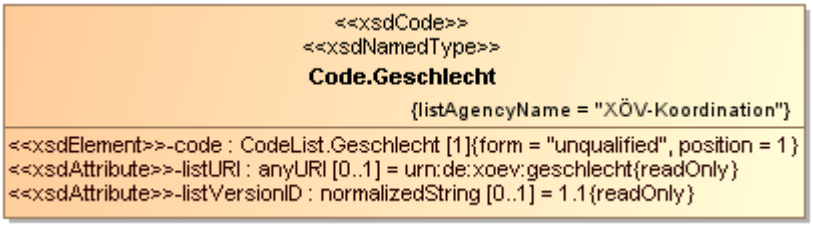
8.3.6.3 Modellierung von Codelisten-Einträgen

Eine UML-Enumeration zu einer Codeliste besteht aus mehreren UML-Enumerations-Literalen. Jedem dieser UML-Enumerations-Literale ist der Stereotyp `xsdCodeListEntry` zugewiesen.

Über die Eigenschaft "name" dieses Stereotyps wird der Codename des Codelisten-Eintrags festgelegt. Bspw. ist "Schönheitswettbewerb" der Codename für den Code 1.

8.3.6.4 Wiederverwendung von XÖV-Codelisten

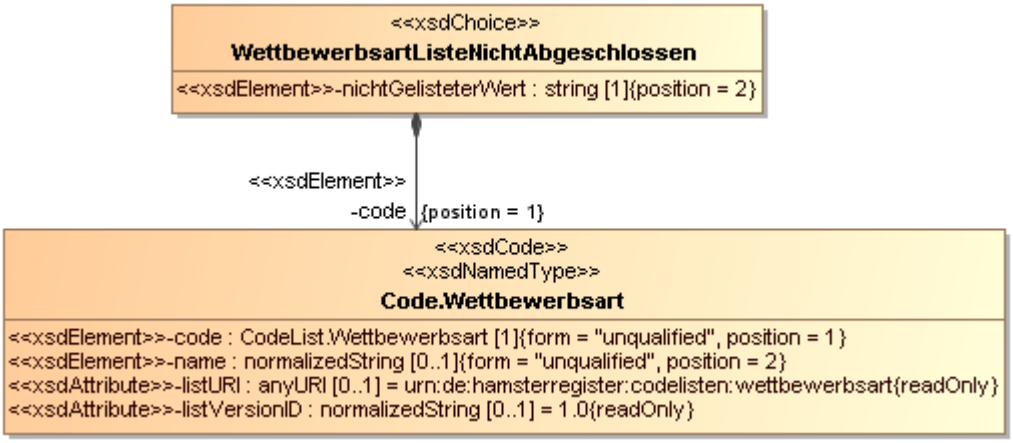
In XHamsterzucht wird das Geschlecht eines Hamsters mit den von der XÖV-Koordination vorgegebenen Codelisteneinträgen als Standard Codeliste (Typ 1) abgebildet.



Grundsätzlich müssen alle wiederverwendeten XÖV-Codelisten eins-zu-eins übernommen werden. Die beiden UML-Enumerations-Literale der UML-Enumeration "CodeList.Geschlecht" entsprechenden den Vorgaben der XÖV-Codeliste. Bei der UML-Klasse des Datentyps "Code.Geschlecht" ist in der Stereotyp-Eigenschaft "listAgencyName" die "XÖV-Koordination" als Herausgeber dokumentiert.

8.3.6.5 Modellierung nicht abgeschlossener Codelisten

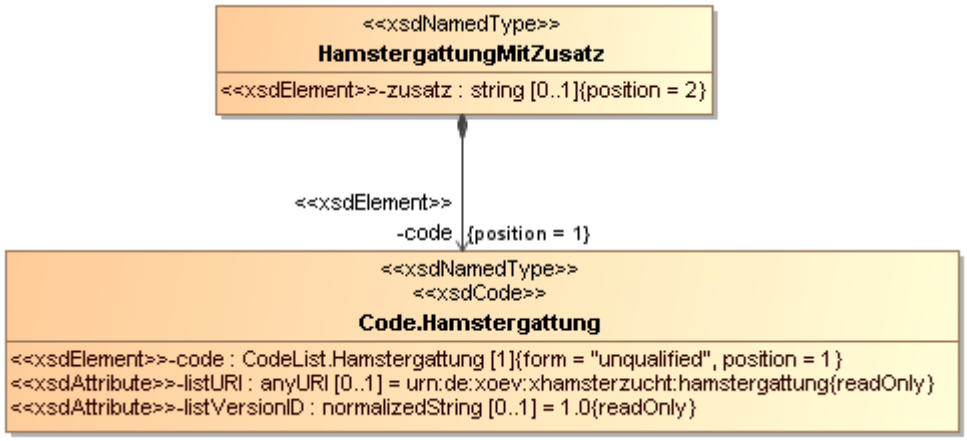
Die Einbindung der Codeliste für die Wettbewerbsart ist als nicht abgeschlossene Codeliste modelliert. d.h. es ist nicht auszuschließen, dass auch andere Wettbewerbsarten gemeldet werden, die noch nicht in der Codeliste erfasst sind:



Die UML-Klasse "Wettbewerbsart_ListeNichtAbgeschlossen" ist mit dem Stereotyp `xsdChoice` versehen. Sie referenziert die UML-Klasse des Code-Datentyps "Code.Wettbewerbsart" und definiert die Eigenschaft "nichtGelisteterWert". Dadurch ist es möglich, in einer XML-Instanz einer Nachricht entweder einen Codelisten-Eintrag oder einen nicht in der Codeliste vorkommenden Wert zu übermitteln.

8.3.6.6 Modellierung unscharfer Codelisten

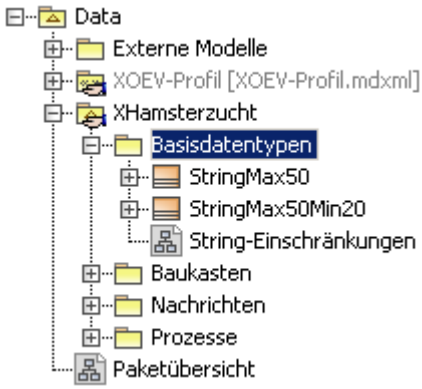
Die Einbindung der Codeliste für die Hamstergattung ist als unscharfe Codeliste modelliert. d.h. ein ausgewählter Eintrag wird noch näher beschrieben:



Die UML-Klasse "Hamstergattung_MitZusatz" referenziert die UML-Klasse des Code-Datentyps "Code.Hamstergattung" und definiert das optionale Element "zusatz". Dadurch ist es möglich, in einer XML-Instanz einer Nachricht die Auswahl eines Codelisten-Eintrags zur Hamstergattung durch die Angabe eines zusätzlichen Hinweises weiter zu präzisieren.

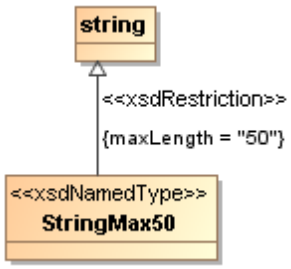
8.3.7 Abbildung von Basisdatentypen

Prinzipiell bilden die XÖV-Basisdatentypen die Grundlage für die als fachliche Datentypen abgebildeten UML-Klassen im Paket "Baukasten" bzw. "Nachrichten". Da die durch die XÖV-Basisdatentypen vorgegebenen Datentypen nicht gänzlich für die Modellierung der fachlichen Anforderungen von XHamsterzucht ausreichen, wurden zusätzliche Basisdatentypen in dem Projekt angelegt. Diese Datentypen befinden sich unterhalb des Pakets "Basisdatentypen":

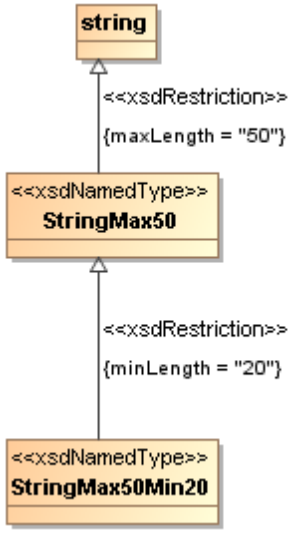


Für die Namensgebung und die Modellierung der Basisdatentypen sind die gleichen Regeln zu beachten, wie für die Erstellung der fachlichen Datentypen. Im Unterschied zu diesen arbeitet man jedoch hier häufig mit der Restriktion bestehender XÖV-Basisdatentypen statt mit deren Komposition. In XHamsterzucht wurden die beiden Basisdatentypen "StringMax50" und "StringMax50Min20" als UML-Klassen angelegt.

Beide Klassen sind vom XÖV-Basisdatentyp "string" abgeleitet. Die Ableitungen sind im UML-Modell als UML-Abhängigkeiten mit dem Stereotyp `xsdRestriction` abgebildet. Je nach Beschaffenheit des Datentyps wurden unterschiedliche Stereotyp-Eigenschaften mit Werten für XML-Facetten belegt:



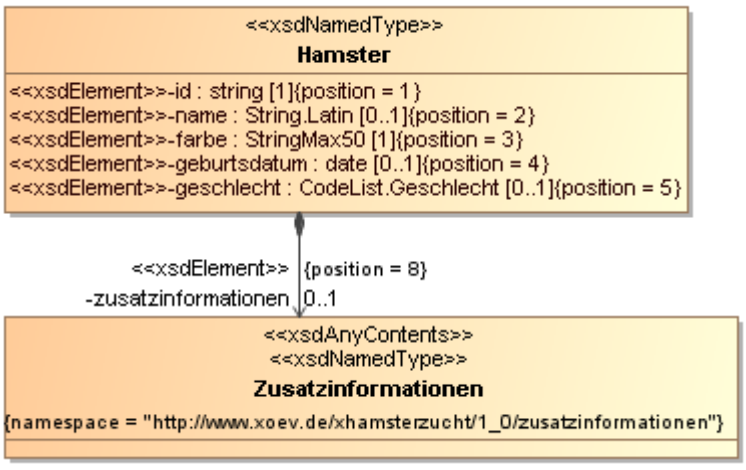
Für die UML-Klasse des Basisdatentyps "StringMax50" wurde die Stereotyp-Eigenschaft "maxLength" mit dem Wert "50" belegt. Dadurch muss in einer XML-Instanz dieses Basisdatentyps eine Zeichenkette mit einer maximalen Anzahl von 50 Zeichen übermittelt werden.



Die UML-Klasse des Basisdatentyps "StringMax50Min20" ist eine Ableitung der UML-Klasse "StringMax50". Daher gilt auch für sie die oben genannte Restriktion (maxLength = 50). Zusätzlich zu dieser Restriktion wurde die Stereotyp-Eigenschaft "minLength" mit dem Wert "20" belegt. Dadurch muss in einer XML-Instanz dieses Basisdatentyps eine Zeichenkette mit einer maximalen Anzahl von 50 und einer minimalen Anzahl von 20 Zeichen übermittelt werden.

8.3.8 Darstellung von Wildcard-Elementen

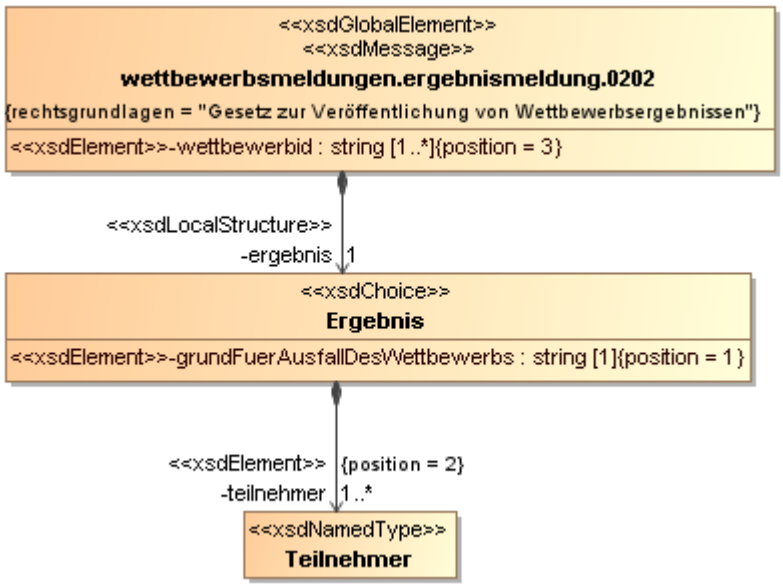
Die UML-Klasse des fachlichen Datentyps "Zusatzinformationen" wurde als XML-Wildcard-Element modelliert:



Zusätzlich ist ihr der Namensraum "http://www.xoev.de/xhamsterzucht/1_0/zusatzinformationen" über die Eigenschaft "namespace" des Stereotyps `xsdAnyContents` zugewiesen. Auf diese Weise wird der fachliche Datentyp im XML-Schema als Typ umgesetzt, der beliebige XML-Elemente eines bestimmten Namensraumes enthalten kann.

8.3.9 Vereinfachte Darstellung von XML-Instanz-Inhalten

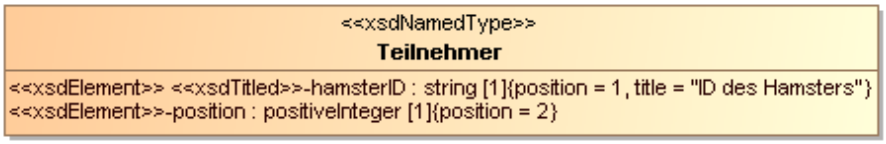
Tief verschachtelte komplexe Datentypen machen die Erstellung einer XML-Instanz dieser Typen zunehmend aufwändig. Dies ist häufig bei Datentypen der Fall, deren UML-Klasse mit dem Stereotyp `xsdChoice` versehen wurde.



Die Mitteilung der Wettbewerbsergebnisse lässt die Übermittlung eines echtes Ergebnisses oder eines Grundes für den Ausfall des Wettbewerbs zu. Im UML-Modell wurde daher zwischen den UML-Klassen des Nachrichten-Typs "wettbewerbsmeldungen.ergebnismeldung.0202" und des fachlichen Datentyps "Ergebnis" eine Assoziation mit dem Namen "ergebnis" modelliert. Diese Assoziation wurde mit dem Stereotyp `xsdLocalStructure` versehen. Dies hat zur Folge, dass sich der fachliche Datentyp "Ergebnis" im Schema als anonymer komplexer Typ direkt unterhalb des Nachrichten-Typs "wettbewerbsmeldungen.ergebnismeldung.0202" befindet. In einer XML-Instanz dieser Nachricht wird daher neben dem Element "wettbewerbid" entweder das Element "grundFuerAusfallDesWettbewerbs" oder mehrere Elemente "teilnehmer" angegeben. Die Zwischenebene "ergebnis" wird ausgelassen.

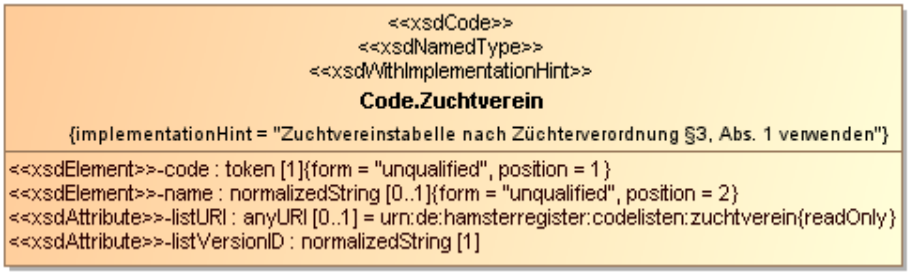
8.3.10 Ergänzungen zur Dokumentation

Aus Abschnitt 8.3.1 auf Seite 105 gehen Anweisungen zur grundlegenden Benennung und Dokumentation eines XÖV-Standards hervor. Häufig müssen UML-Klassen und deren Eigenschaften aufgrund der technischen Umsetzbarkeit anders benannt werden als man dies in einem fortlaufenden Text tun würde. Um die Lesbarkeit der Dokumentation dennoch zu erhöhen, können für diese Elemente sprechende Namen vergeben werden, die in der Dokumentation an die Stelle der technischen Namen treten.



Die UML-Klasse "Teilnehmer" enthält die Eigenschaft "hamsterID". Die Stereotyp-Eigenschaft "title" (Stereotyp `xsdTitled`) legt fest, dass die Eigenschaft "hamsterID" in der Dokumentation den Titel "ID des Hamsters" erhält.

Weiterhin kann die Dokumentation um Implementierungshinweise ergänzt werden, die für die Verfahrenshersteller von entscheidender Bedeutung sind. Ein Beispiel hierfür sind die Dokumentationen der Code-Datentypen 2 - 4.



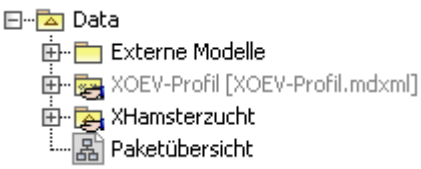
Bei der UML-Klasse "Code.Zuchtverein" handelt es sich um eine versionsfreie Codeliste (Typ 3). Sie wurde mit dem Stereotyp `xsdWithImplementationHint` versehen. Die Stereotyp-Eigenschaft "implementationHint" enthält den Hinweis, dass für die Übertragung eines Wertes zu dieser Codeliste die Zuchtvereinstabelle nach Züchterverordnung §3, Abs. 1 verwendet werden soll.

8.3.11 Übersicht der Pakete

Für die korrekte Verarbeitung des UML-Modells durch das XÖV-Produktionszubehör (i.e.S. der XGenerator sowie XÖV-Invarianten, XÖV-XSD-Vorlagen und XÖV-XSD-Operationen) müssen die mit dem Stereotyp `xsdSchema` gekennzeichneten Pakete des UML-Modells in einer Paketübersicht mit UML-Abhängigkeiten zueinander in Beziehung gesetzt werden.

Hinweis: Bei komplexen Datenmodellen mit vielen Paketen bietet es sich an, diese Zusammenhänge in mehrere Paketübersichten aufzuteilen. In jeder Übersicht sollte dann nur ein UML-Paket mit seinen Abhängigkeiten zu anderen Paketen im Fokus stehen.

Das Datenmodell von XHamsterzucht ist einfach strukturiert, so dass eine zentrale Paketübersicht ausreichend ist. Die Paketübersicht befindet sich auf oberster Ebene:



Hinweis: Die Übersicht könnte auch unterhalb des Modells XHamsterzucht abgelegt sein. Es ist lediglich zu beachten, dass sie zentral abgelegt ist und durch einen Modellierer leicht gefunden werden kann.

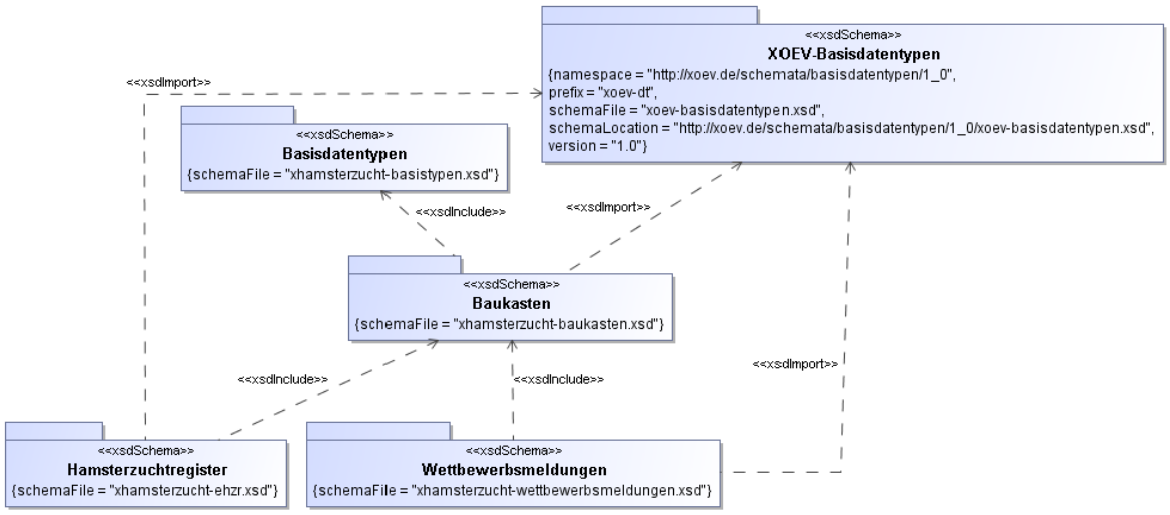
Bei der Erstellung von Paketübersichten sind folgende Regeln zu beachten:

- Namen von XML-Schema-Dateien (siehe NDR-18 auf Seite 72)
- Physische Speicherorte von XML-Schemata als URL (siehe NDR-26 auf Seite 75)
- Verwendung von Original-Namespace-Präfixen bei Schema-Importen (siehe NDR-27 auf Seite 75)
- Valide W3C-XML-Schemata (siehe NDR-28 auf Seite 76)
- Identifizierende Namensräume (siehe NDR-29 auf Seite 76)
- Versionierung der Schemata (siehe NDR-30 auf Seite 76)
- Namensräume mit Versionen (siehe NDR-31 auf Seite 77)

Die folgenden Abschnitte liefern eine Beschreibung zur Umsetzung der benannten Regeln.

8.3.11.1 Grundsätzliche Modellierung

In der Paketübersicht sind die UML-Pakete "Basisdatentypen", "Baukasten", "Wettbewerbsmeldungen" und "Hamsterzuchtregister" und die Pakete des externen Modells "XOEV-Basisdatentypen" miteinander über UML-Abhängigkeiten verknüpft. Diese Abhängigkeiten sind mit den Stereotypen `xsdInclude` bzw. `xsdImport` versehen.



Alle dargestellten Pakete besitzen einen eindeutigen Wert für die Stereotyp-Eigenschaft "schemaFile".

8.3.11.2 implizit gesetzte Werte

Im UML-Modell sind die Stereotyp-Eigenschaften "namespace", "prefix", "schemaLocationBase" und "version" des Stereotyps `xsdXModel` für das Modell "XHamsterzucht" mit den abgebildeten Werten versehen:

Da sich die UML-Pakete "Basisdatentypen", "Baukasten", "Wettbewerbsmeldungen" und "Hamsterzuchtregister" in der Struktur unterhalb des Modells "XHamsterzucht" befinden, sind für sie diese Werte implizit gesetzt. Hierdurch wird jedem der benannten Pakete der eindeutige und mit einer Version versehene Namensraum "http://xoev.de/schemata/xhamsterzucht/1_0" zugewiesen. Weiterhin wird für diese Pakete der Prefix "xhz" und die Version "1.0.0" definiert sowie die Basis für spätere SchemaLocation-Attribute "http://xoev.de/schemata/xhamsterzucht/1_0".

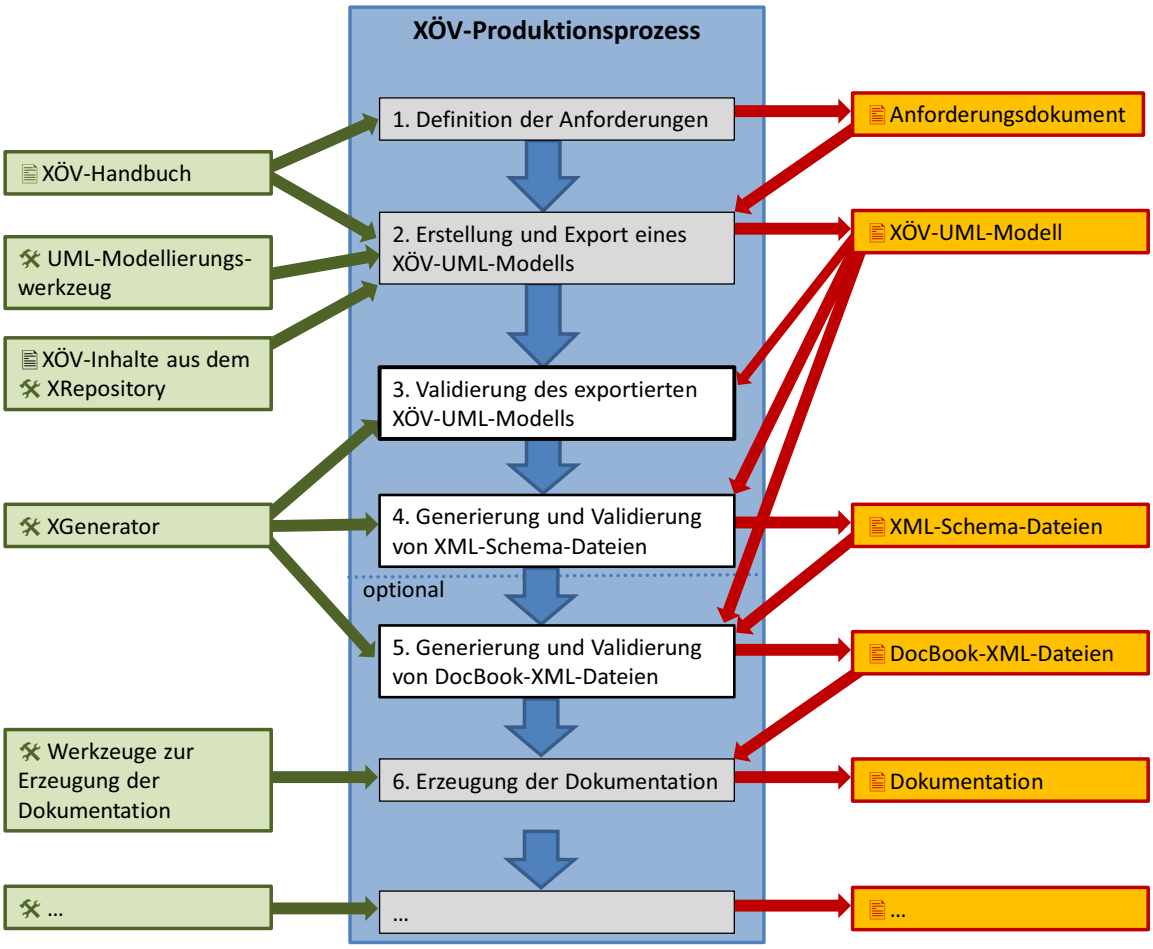
Hinweis: Es ist möglich, diese Werte direkt bei den mit dem Stereotyp `xsdSchema` versehenen Paketen zu hinterlegen. Diese Werte haben dann vor den im `xsdXModel`-UML-Modell eingetragenen Werten Vorrang.

---

9. XGENERATOR

Der XGenerator ist das entscheidende XÖV-Werkzeug, welches bei der Produktion eines XÖV-Standards benötigt wird. Dieses Kapitel soll einen Überblick über die Funktionsweise und das Zusammenspiel des XÖV-Produktionszubehörs geben, welches mit dem XGenerator zum Einsatz kommt. Damit soll ein Grundverständnis über die Zusammenhänge der verschiedenen Bestandteile geschaffen werden (siehe Abschnitt 9.1.1 auf Seite 132), welches bei der Anwendung des XGenerators von Nutzen ist. Im zweiten Teil des Kapitels wird in Form eines Anwenderhandbuchs auf den konkreten Einsatz des XGenerators in der Praxis eingegangen (siehe Abschnitt 9.2 auf Seite 136).

Die folgende Grafik gibt noch einmal einen Überblick über den Produktionsprozess eines XÖV-Standards und zeigt, dass die Schritte 3 bis 5 durch den XGenerator unterstützt werden (siehe auch Abschnitt 3 auf Seite 21).



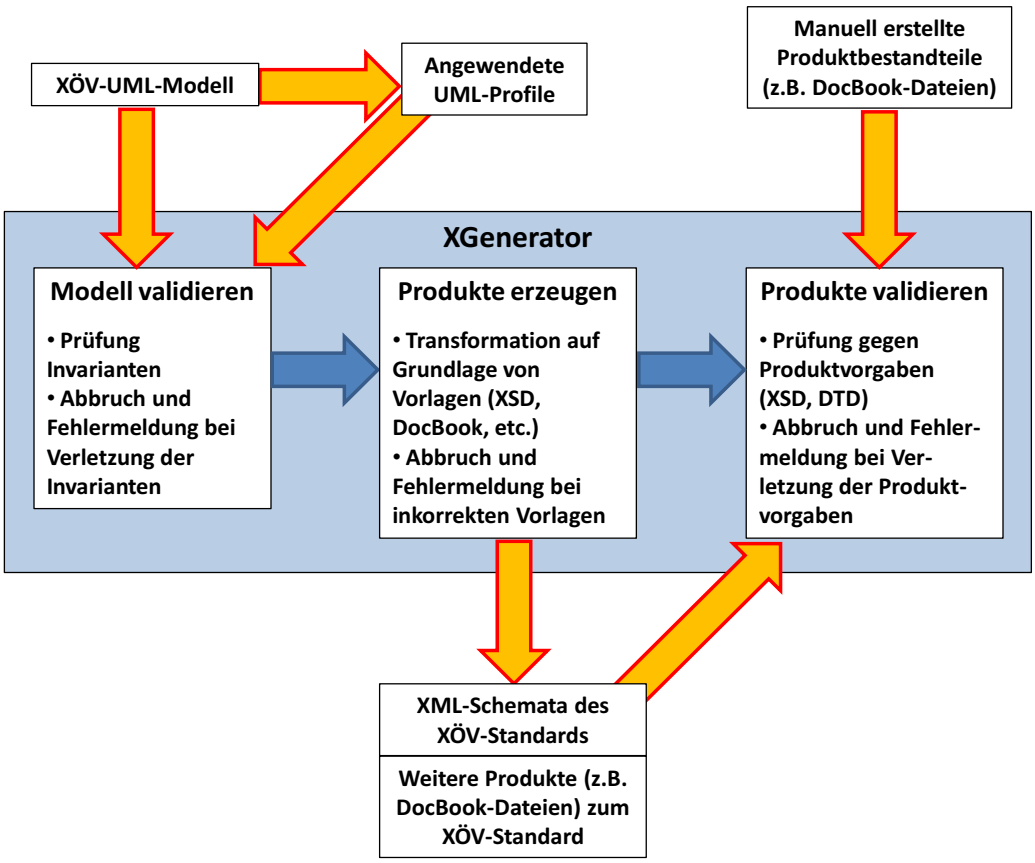
9.1 Der XGenerator und sein Zubehör

Dieser Abschnitt geht auf die einzelnen Bestandteile des XÖV-Produktionszubehörs ein, die eine Relevanz für die Funktionsweise des XGenerators haben. Neben dem Werkzeug selbst, wird erläutert, welche Rolle dieses Zubehör (Invarianten, Operationen und Vorlagen) bei der Arbeit mit dem XGenerator spielt.

9.1.1 XGenerator

Der XGenerator an sich, ist ein sehr generisches Werkzeug. Im Kontext von XÖV unterstützt er die XÖV-Modellierer bei der Produktion eines XÖV-Standards. Aus dem XÖV-UML-Modell können mittels XGenerators die für einen XÖV-Standard benötigenten XML-Schema-Dateien generiert werden. Darüber hinaus unterstützt das Werkzeug bei Bedarf auch die automatisierte Erzeugung einer Dokumentation dieser XML-Schema-Dateien. Auf diese Weise ist sichergestellt, dass die Dokumentation des XÖV-Standards konsistent zu den generierten XML-Schema-Dateien ist. Gerade bei der Pflege eines XÖV-Standards ist es von Vorteil, alle relevanten Informationen an einer Stelle, im XÖV-UML-Modell zu pflegen und aus diesem die weiteren Bestandteile des Standards automatisiert zu generieren.

Die nachfolgende Grafik zeigt die wesentlichen Funktionen des XGenerators: Validierung des Modells, Generierung von Produkten und deren Validierung im Kontext von XÖV.



Um im UML-Modell die notwendigen Informationen zur Datenübertragung mit XML abzulegen, wird das UML-Modell gemäß der XÖV-Vorgaben mit dem XÖV-UML-Profil profiliert (siehe Abschnitt Abschnitt 4 auf Seite 25) und so zu einem XÖV-UML-Modell. Diese Zusatzinformationen, im UML-Modell dargestellt

durch Stereotypen, sind technische Details für die Erzeugung der XML-Schema-Dateien. Das XÖV-UML-Profil definiert darüber hinaus bestimmte Regeln (Invarianten), die festlegen, wie die Stereotypen aus dem XÖV-UML-Profil anzuwenden sind (siehe Abschnitt Abschnitt 5 auf Seite 61).

Nachdem das XÖV-UML-Modell und alle angewendeten UML-Profile als XMI-Export in den XGenerator eingelesen wurden, findet in einem ersten Schritt eine Validierung dieses Modells hinsichtlich der korrekten Anwendung des XÖV-UML-Profils auf Basis der verwendeten Stereotypen und Invarianten statt (siehe auch Abschnitt 4 auf Seite 25). Bei einem Verstoß wird keine Generierung durchgeführt. Stattdessen wird der Modellierer darüber informiert, welche Teile des XÖV-UML-Modells gegen welche Regeln aus dem XÖV-UML-Profil verstoßen (siehe Abschnitt 9.2.3.6 auf Seite 152).

Nach einer erfolgreichen Validierung folgt als nächster Schritt die Generierung von XML-Schema-Dateien aus dem eingelesenen XÖV-UML-Modell. Diese erzeugten Dateien werden auf valides XML-Schema geprüft. In einem letzten Schritt werden ebenfalls auf der Grundlage des XÖV-UML-Modells, aber auch auf der Basis der generierten XML-Schema-Dateien docbook-Dateien und SVG-Grafiken erzeugt, die in eine umfassende Dokumentation (mit manuell erstellten Teilen) des XÖV-Standards eingebunden werden können. Ob die Einbindung in die auf DocBook basierende Gesamtdokumentation erfolgreich war, wird ebenfalls vom XGenerator überprüft. Darüber hinaus könnten optional weitere XML-Artefakte vom XGenerator erzeugt werden. In Arbeit ist die Generierung von WSDL-Dateien.

Weitere Details zur praktischen Nutzung des XGenerators im Rahmen eines XÖV-Vorhabens sind in Abschnitt 9.2 auf Seite 136 zu finden.

9.1.2 XGenerator-Zubehör: Invarianten, Vorlagen und Operationen

In diesem Abschnitt sollen Hintergrundinformationen zu Invarianten, Vorlagen und Operation gegeben werden. Grundsätzlich benötigen XÖV-Modellierer aber kein tiefer gehendes Wissen in diesem Bereich.

Wie bereits erwähnt, ist der XGenerator ein sehr generisches Werkzeug, d.h. im Allgemeinen können beliebige Profile verwendet und somit unterschiedliche Invarianten geprüft werden, da das XÖV-UML-Profil nicht fest im XGenerator verortet ist (siehe Abschnitt 9.1.2.1 auf Seite 134). In Abhängigkeit von dem verwendeten UML-Profils können dann bestimmte XML-Dateien generiert werden. Hierfür werden Vorlagen und Operationen benötigt, um die entsprechenden, durch ein UML-Profil gekennzeichnete Informationen aus dem UML-Modell herauszulesen und als XML-Dateien herauszuschreiben. Somit sind Invarianten, Vorlagen und Operationen für die Funktionsweise des XGenerators unerlässlich. Also nur unter Verwendung solcher Dateien validiert und transformiert der XGenerator ein eingelesenes UML-Modell.

Für die XÖV-Zwecke wurden spezifische Dateien erstellt, die genau die, in diesem XÖV-Handbuch beschriebenen, Vorgaben umsetzten (siehe Abschnitt 5 auf Seite 61). Auf diese Weise ist sichergestellt, dass aus einem XÖV-UML-Modell XÖV-konforme Ergebnisse, also XÖV-Standards generiert werden. Für die Modellierer von XÖV-Standards besteht in der Regel also kein Bedarf, die genannten Dateien für den XGenerator zu verändern. Im Gegenteil, bei Veränderungen gehen ggf. für die XÖV-Konformität relevante Inhalte verloren. (Ausnahme: XÖV-DocBook-Musteroperationen, hier können standardspezifische Anpassungen vorgenommen werden.)

Unter http://www.xoev.de sind im XÖV-Kontext verfügbar:

- XÖV-XSD-Vorlagen
- XÖV-DocBook-Mustervorlagen

- XÖV-Invarianten (unterschieden in Muss, Soll und Empfehlungen)
- XÖV-XSD-Operationen
- XÖV-DocBook-Musteroperationen

Die Invarianten, Vorlagen und Operationen beinhalten bezüglich ihres Aufbaus Aspekte aus zwei verschiedenen Bereichen:

1. Velocity-Syntax für die Steuerung des Ablaufs in den Vorlagen
2. OCL-Abfragen auf Inhalte im UML-Modell für Vorlagen, Invarianten und Operationen

Allgemeine Informationen zu den Themen "Velocity" und "OCL" stehen im Anhang Abschnitt B auf Seite 157 zur Verfügung.

9.1.2.1 Die .invariants-Dateien (Invarianten)

Diese Dateien enthalten die Regeln (=Invarianten) für z.B. das/die anzuwendende(n) Profile(e) (siehe Abschnitt 5 auf Seite 61). Alle .invariants-Dateien, die sich im `model`-Verzeichnis des XGenerator befinden, werden automatisch geladen und geprüft (siehe Abschnitt 9.2.1.3.1 auf Seite 138).

Im Rahmen des XÖV-Produktionszubehörs sind die folgenden Invarianten-Dateien definiert:

Invarianten-Datei	Inhalt und Verwendung
NDR-MUSS.invariants	Invarianten zur Validierung des Modells im XGenerator mit Bezug zu den MUSS-Namens- und Entwurfsregeln aus dem XÖV-Handbuch
NDR-SOLL.invariants	Invarianten zur Validierung des Modells im XGenerator mit Bezug zu den SOLL-Namens- und Entwurfsregeln aus dem XÖV-Handbuch
NDR-EMPFEHLUNG.invariants	Invarianten zur Validierung des Modells im XGenerator mit Bezug zu den Empfehlungen der Namens- und Entwurfsregeln aus dem XÖV-Handbuch

---

Hinweis: Die XÖV-Invarianten zu den MUSS-Regeln dieses Handbuchs müssen im Rahmen der XÖV-Konformität unverändert übernommen werden. Die Invarianten zu den SOLL-Regeln müssen spezifisch für jedes XÖV-Vorhaben auf ihren Einsatz hin geprüft werden. Die Invarianten zu den Empfehlungen können, müssen jedoch nicht verwendet werden.

Jede invariants-Datei besteht zwingend aus einem <invariants>-Wurzelelement mit einem Namensraum "www.osci.de/xoev/invariants". In dem Wurzelelement können beliebig viele <invariant>-Elemente enthalten sein. Jedes <invariant>-Element definiert genau eine Well-formedness Rule. Dabei sind die XML-Attribute 'context' (die Metaklasse, auf die sich diese Regel bezieht) und 'name' (der Name dieser Regel) zwingend anzugeben.

Als Unterelemente von <invariant> sind ein <body>-Element (die OCL-Definition der Invariante) und ein optionales <documentation>-Element anzugeben. Die Dokumentation wird, falls vorhanden, im XGenerator mit angezeigt. Der OCL-Ausdruck im <body>-Element muss ein boolscher Ausdruck sein.

Beispiel für eine Invariante:

```
<invariant context='Class' name='MessagesMustBeGlobalElements'>
  <body>
    self.extension_xsdMessage.isDefined() and self.hlpIsPartOfXModel() implies
      self.extension_xsdGlobalElement.isDefined()
  </body>
  <documentation>Nachrichten müssen gloable Elemente sein.</documentation>
</invariant>
```

Hinweis: Invarianten, die nicht für die XÖV-Konformität erforderlich sind, können aus der Validierung durch den XGenerator ausgeschlossen werden, in dem die Syntax "<!−−" und "−−>" verwendet wird:

9.1.2.2 Die .vm-Dateien (Vorlagen)

Bei der Erzeugung von XML-Schema-Dateien, DocBook-Dateien oder ggf. auch anderen Output-Dateien arbeitet der XGenerator mit Template-Dateien. Die Vorlagen sind in der Sprache *Velocity* (Dateiendung *.vm) erstellt und enthalten OCL-Aufrufe zur Navigation und Auswertung der Modelle (siehe Abschnitt B auf Seite 157).

Hinweis: Die durch die XÖV-Koordination zur Verfügung gestellten Vorlagen zur Generierung der XML-Schema-Dateien (XÖV-XSD-Vorlagen) dürfen im Zuge der XÖV-Konformität nicht geändert werden, da in diesen Vorgaben aus den XÖV-Namens- und Entwurfsregeln (siehe Abschnitt 5 auf Seite 61 umgesetzt sind.

9.1.2.3 Die Operationen-Dateien (.operations)

Operationen-Dateien enthalten häufig genutzte Operationen in Form von OCL-Abfragen, die im Rahmen der Invarianten und Vorlagen verwendet werden. Diese Hilfsoperationen sind damit an allen Stellen, an denen OCL ausgewertet wird, verfügbar. Alle .operations-Dateien, die sich im model-Verzeichnis des XGenerator-Projekts befinden, werden automatisch mit geladen (siehe Abschnitt 9.2.1.3.1 auf Seite 138).

Im Rahmen des XÖV-Produktionszubehörs sind die folgenden XÖV-XSD-Operationen und XÖV-DocBook-Musteroperationen definiert:

Operationen-Dateien	Inhalt und Verwendung
XSD.operations	Operationen (OCL-Aufrufe), die in den XÖV-Invarianten, XÖV-XSD-Vorlagen sowie XÖV-DocBook-Mustervorlagen wiederverwendet werden
DocBook.operations	Operationen (OCL-Aufrufe), die in den XÖV-DocBook-Mustervorlagen wiederverwendet werden

---

Hinweis: Die Operationen, die im Rahmen der XÖV-XSD-Operationen zur Verfügung gestellt werden, dürfen zur Erreichung der XÖV-Konformität nicht verändert werden.

Jede operations-Datei besteht zwingend aus einem <additionalOperations>-Wurzelelement:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <additionalOperations xmlns="http://www.osci.de/xoev/operations">
```

Der Namespace ist dabei relevant und wie oben zu setzen. In dem Wurzelelement können beliebig viele <additionalOperation>-Elemente enthalten sein. Jedes <additionalOperation>-Element definiert eine Operation. Dabei sind die XML-Attribute 'context' (die Metaklasse, für die die Operation definiert werden soll) und 'name' (der Name der Operation) zwingend anzugeben. Als Unterelemente sind ein <result>-Element mit dem OCL-Rückgabetyp der Operation ('type'-Attribut) und, wie bei den Invarianten, ein <body>-Element und ein optionales <documentation>-Element anzugeben.

```
<operation context='Element' name='hlpOwningPackage'>
  <result type='Package'/>
  <body>
    if self.owner.oclIsTypeOf(Class) then
      self.owner.hlpOwningPackage()
    else
      self.owner.oclAsType(Package)
    endif
  </body>
  <documentation>Das Paket, in welchem dieses Element enthalten ist. Bei Elementen
in inneren Klassen wird aufsteigend nach einem Paket gesucht.</documentation>
</operation>
```

9.2 Anwendung des XGenerators

Dieser Abschnitt ist als Anwenderhandbuch für den Einsatz des XGenerators gedacht.

Der XGenerator ist für den XÖV-Kontext unter http://www.xoev.de verfügbar und außerdem inklusive seiner Source-Code-Dateien unter https://joinup.ec.europa.eu/software/xgenerator/home.

9.2.1 Vorbereitung für die Anwendung des XGenerators

Bevor mit dem XGenerator gearbeitet werden kann, müssen bestimmte Vorbereitungen getroffen werden. Neben der eigentliche Installation gehört die Erstellung eines "Projektverzeichnisses" dazu, in dem die durch den XGenerator einzulesenden Dateien abzulegen sind. Außerdem wird auf bestimmte Konfigurationsmöglichkeiten eingegangen.

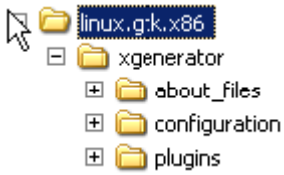
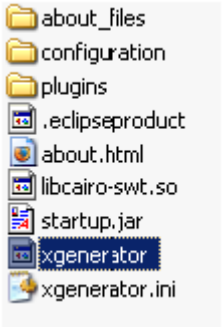
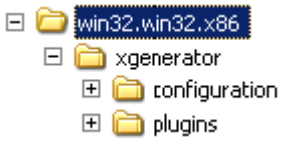
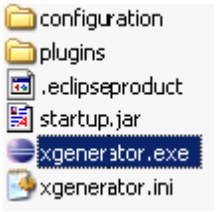
9.2.1.1 Voraussetzungen für den XGenerator

Die technischen Voraussetzungen zur Inbetriebnahme des XGenerators beschränken sich auf die in der unten stehenden Tabelle angegebenen Inhalte.

Kriterium	Voraussetzung
Betriebssystem	Windows, Linux
Weitere Software	Java ab Version 1.5 (Java 5) muss installiert sein
Hardware	Arbeitsplatzrechner, 1 GB RAM, keine Netzwerkverbindung notwendig

9.2.1.2 Installation des XGenerator

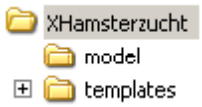
Der XGeneratorsteht für die Betriebssysteme Windows bzw. Linux zur Verfügung als Produktverzeichnis zur Verfügung[1]. Dieses Verzeichnis wird für die Inbetriebnahme des XÖV-Werkzeugs von einer Installations-CD bzw. von einer Website an eine beliebige Stelle der lokalen Festplatte übertragen. Die folgende Tabelle zeigt den Aufbau des Produktverzeichnisses für die beiden Betriebssysteme und des darin enthaltenen xgenerator-Verzeichnisses. Die markierten xgenerator-Dateien dienen dem Aufruf des XGenerators.

Betriebssystem	Produktverzeichnis	xgenerator-Verzeichnis
Linux Rootverzeichnis: linux.gtk.x86 Startroutine im Verzeichnis: xgenerator		
Windows Rootverzeichnis: win32.win32.x86 Startroutine im Verzeichnis: xgenerator.exe		

Mit der Ausführung der Startroutine "xgenerator.exe" bzw. "xgenerator" aus dem gleichnamigen Verzeichnis kann der XGenerator als Werkzeug mit Benutzeroberfläche gestartet werden.[2]

9.2.1.3 Input-Verzeichnisstruktur für Invarianten, Vorlagen und Operationen

Bevor ein Projekt mit dem XGenerator geöffnet bzw. bearbeitet werden kann, muss ein Projektverzeichnis (z.B. "XHamsterzucht") angelegt sein, welches mindestens die beiden Unterverzeichnisse "templates" und "model" enthält. Die Namen dieser beiden Unterverzeichnisse werden genau so vom XGenerator benötigt, dürfen also nicht geändert werden. Das Projektverzeichnis kann lokal an beliebiger Stelle abgelegt werden. Die nachfolgende Abbildung zeigt ein Beispiel-Verzeichnis.



1.Details siehe technische Dokumentation zum XGenerator: https://joinup.ec.europa.eu/software/xgenerator/home

2.Eine Ausführung über Kommando-Zeile ist ebenfalls möglich.

Zusätzlich können auch weitere Unterverzeichnisse angelegt werden, die für das Projekt, aber nicht für die Anwendung des XGenerators von Bedeutung sind.

Sollen mehrere Projekte mit dem XGenerator bearbeitet werden können, so muss für jedes Projekt ein separates Projektverzeichnis angelegt werden, mit jeweils einem eigenen model- und templates-Ordner, auch wenn es ggf. projektübergreifend in diesen Unterverzeichnissen redundante Dateien gibt.

9.2.1.3.1 Verzeichnis "model"

Das model-Verzeichnis enthält alle Inputfiles für den XGenerator. Das sind einerseits die Dateien, die unmittelbar mit dem UML-Modell und den angewandten UML-Profilen zusammenhängen (aus dem UML-Case-Tool exportiert), und andererseits die im XGenerator benötigten Dateien zur Validierung bzw. Templateauswertung des Modells (Invarianten und Operationen, siehe Abschnitt 9.1.2 auf Seite 133). Die folgende Tabelle erläutert, welche Dateien für ein XÖV-Vorhaben vom XGenerator benötigt und eingelesen werden.

Datei- bzw. Verzeichnisname	Inhalt und Verwendung
XHamsterzucht.uml	UML-Modell (hier von XHamsterzucht)
*.profile.uml	im UML-Modell genutzte UML-Profile
XSD.operations	XÖV-XSD-Operationen (OCL-Aufrufe), die in den XÖV-Invarianten, XÖV-XSD-Vorlagen sowie XÖV-DocBook-Mustervorlagen wieder verwendet werden
DocBook.operations	XÖV-DocBook-Musteroperationen (OCL-Aufrufe), die in den XÖV-DocBook-Mustervorlagen wiederverwendet werden
NDR-MUSS.invariants	Teil der XÖV-Invarianten zur Validierung des Modells im XGenerator mit Bezug zu den MUSS-Namens- und Entwurfsregeln aus dem XÖV-Handbuch
NDR-SOLL.invariants	Teil der XÖV-Invarianten zur Validierung des Modells im XGenerator mit Bezug zu den SOLL-Namens- und Entwurfsregeln aus dem XÖV-Handbuch
NDR-EMPFEHLUNG.invariants	Teil der XÖV-Invarianten zur Validierung des Modells im XGenerator mit Bezug zu den Empfehlungen der Namens- und Entwurfsregeln aus dem XÖV-Handbuch

Der XGenerator kann UML 2.0-Modelle und -Profile einlesen, die das Format EMF-XMI (XMI 2.X) haben.

Es ist darauf zu achten, dass der model-Ordner nur ein Fachmodell mit zugehörigen Profilen enthält. Bei einem erneuten Export eines UML-Modells muss dieses gemeinsam mit den beim Export aktuell generierten Profilen in den model-Ordner eingestellt werden und alte Modell- bzw. Profildateien sollten gelöscht bzw. überschrieben werden. Alte Profile führen ggf. zu Fehlermeldungen im XGenerator (siehe Abschnitt 9.2.3.6 auf Seite 152).

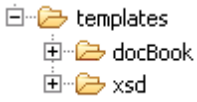
Neben dem bereits benannten XÖV-UML-Profil kann der XGenerator auch Modelle mit anderen Profilen einlesen.

Die Invarianten- und Operationen-Dateien müssen einem Schema genügen (Abschnitt 9.1.2 auf Seite 133). Durch die Eintragung des namespaces in diese Dateien erkennt der XGenerator die Schema-Dateien, auch ohne eine Verbindung ins WWW, da diese Schema-Dateien zusätzlich in den XGenerator integriert wurden.

Hinweis: Die XÖV-Invarianten zu den MUSS-Regeln dieses Handbuchs sowie die XÖV-XSD-Operationen müssen im Rahmen der XÖV-Konformität unverändert übernommen werden. Die Invarianten zu den SOLL-Regeln müssen spezifisch für jedes XÖV-Vorhaben auf ihren Einsatz hin geprüft werden. Die Invarianten zu den Empfehlungen können, müssen jedoch nicht verwendet werden.

9.2.1.3.2 Verzeichnis "templates"

Die benötigten Vorlagen zur XML-Schema-, zur DocBook- sowie sonstiger Generierung sind in den Unterverzeichnissen "xsd", "docBook", usw. abzulegen.



Die Auslieferung des XGenerators erfolgt mit einem Satz XÖV-konformer Velocity-Vorlagen für die Erzeugung von XML-Schema-Dateien und DocBook. Diese Vorlagen können als Vorlage für die eigene projektspezifische Template-Entwicklung genutzt werden.

Hinweis: Gemäß XÖV-Konformität dürfen die Vorlagen zur XML-Schema-Generierung (XÖV-XSD-Vorlagen) zu keinem Zeitpunkt geändert werden.

9.2.1.4 Festlegung des Zeichensatzes

Die in Abschnitt 9.2.1.3.1 auf Seite 138 beschriebenen Unterverzeichnisse beinhalten jeweils eine Datei "global_variables.vm", in der der Zeichensatz der zu generierenden XML-Dateien festgelegt wird. Es muss immer ein Wert für die Variable "encoding" angegeben werden. Unterstützte Zeichensätze sind ISO-8859-1 und UTF-8.

```
...
#set ($encoding='{encoding[ISO-8859-1|UTF-8]}')
...
```

9.2.1.5 Festlegung zu Pretty-Print

Die in Abschnitt 9.2.1.3.1 auf Seite 138 beschriebenen Unterverzeichnisse beinhalten jeweils eine Datei "global_variables.vm", in der die Pretty-Print-Eigenschaft der zu generierenden XML-Dateien festgelegt wird. Es muss immer ein Wert für die Variable "prettyprint" angegeben werden. Gültige Werte sind "true" oder "false".

```
...
#set ($prettyprint='{false|true}')
...
```

9.2.1.6 Internationalisierung des XGenerator

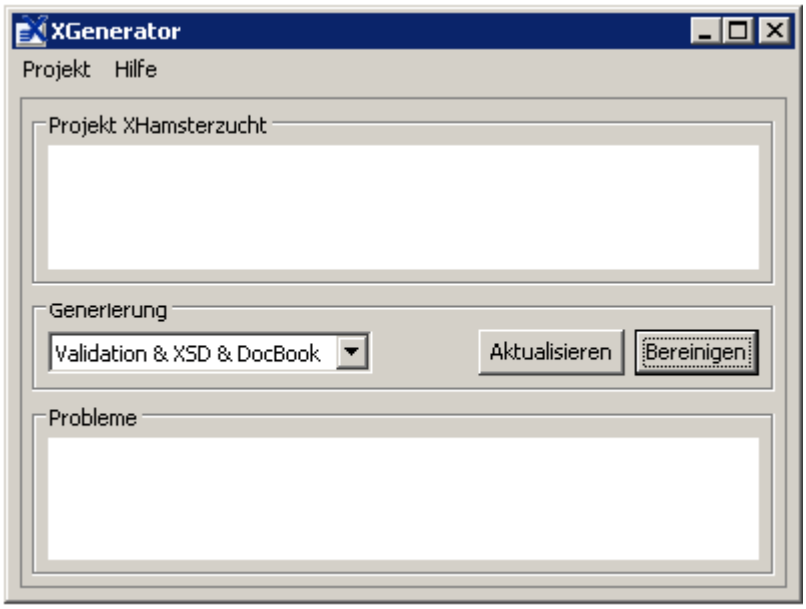
Die Oberfläche des XGenerator kann sowohl in englischer als auch deutscher Sprache genutzt werden. Die Spracheinstellung des Werkzeugs ist an die Spracheinstellung des Betriebssystems gebunden.

9.2.2 Erstellung eines Projekts im XGenerator

Nachdem erklärt wurde, wie für ein XGenerator-Projekt eines XÖV-Standards die Verzeichnisse aufgebaut und welche Dateien enthalten sein müssen, um mit dem XGenerator XML-Schema-Dateien oder auch docbook-Dateien zu erzeugen, wird in diesem Abschnitt beschrieben, wie dieses Projekt im XGenerator-Projekt eingerichtet wird, um die Generierung ausführen zu können.

9.2.2.1 Starten des Werkzeugs

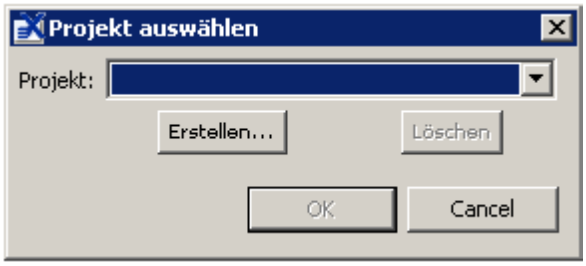
Wie bereits in Abschnitt 9.2.1.2 auf Seite 137 beschrieben, wird das Werkzeug mit dem Aufruf der xgenerator-Datei ("xgenerator.exe" bzw. "xgenerator") gestartet. Nach dem Logo öffnet sich direkt die Benutzeroberfläche des XGenerator.



9.2.2.2 Projekte verwalten

Unter dem Menüpunkt "Verwalten" im Menü "Projekt":

öffnet sich ein Dialog,

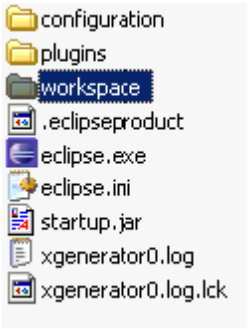


mit dem Projekte

- erstellt,
- geöffnet,
- gewechselt oder
- gelöscht

werden können.

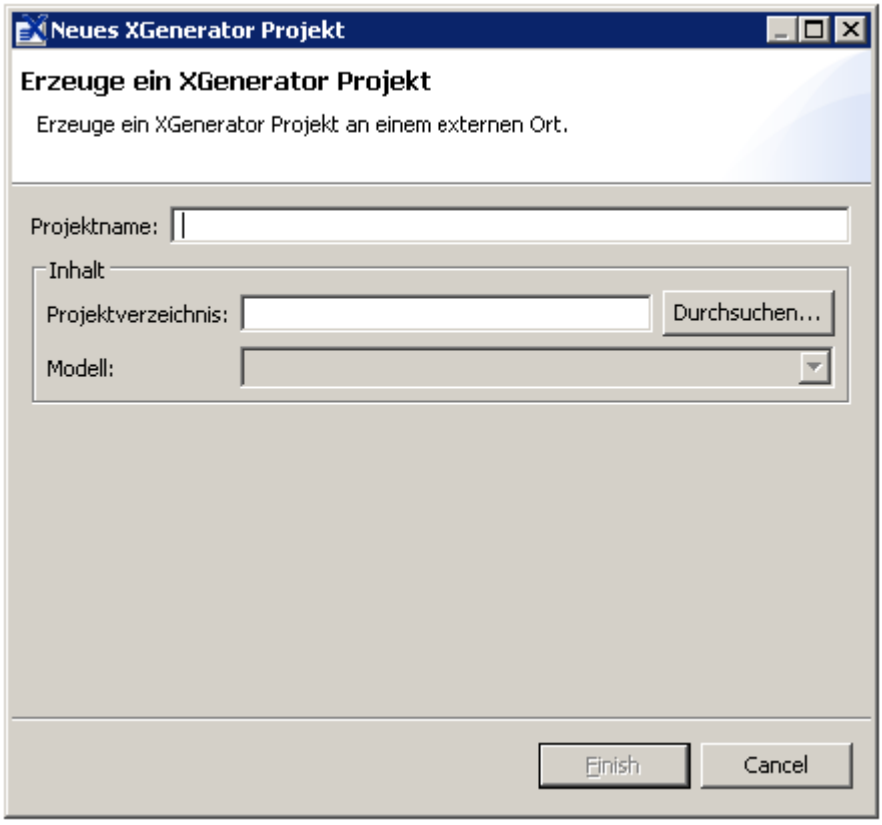
Die Meta-Informationen, welche Projekte wie geöffnet bzw. bearbeitet wurden, werden in einem neu generierten Unterverzeichnis des "Produktverzeichnisses" (siehe Abschnitt 9.2.1.2 auf Seite 137) gespeichert.



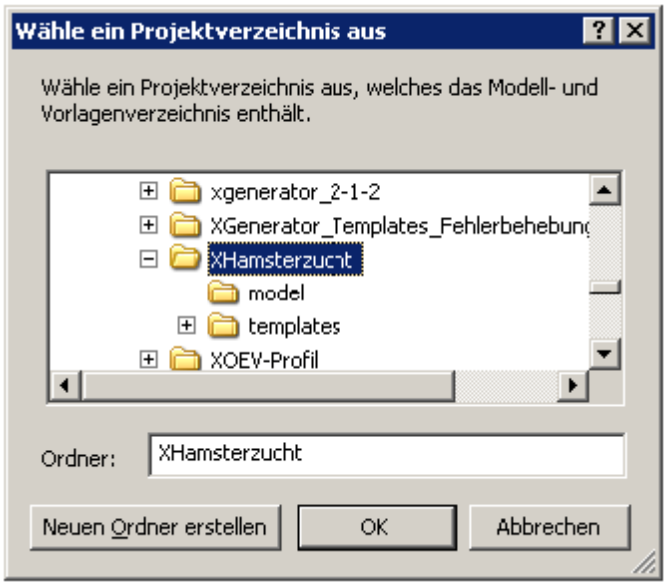
Hinweis: Im Fall einer Fehlermeldung bzgl. hier enthaltener Dateien, kann das Workspace-Verzeichnis gelöscht werden. Die Projekte müssen danach nur erneut im XGenerator, wie im folgenden Abschnitt beschrieben, erstellt werden. Änderungen bzgl. der Generierung von Dateien gehen nicht verloren und werden beim Öffnen des Projektes wie zuvor angezeigt.

9.2.2.3 Projekt erstellen

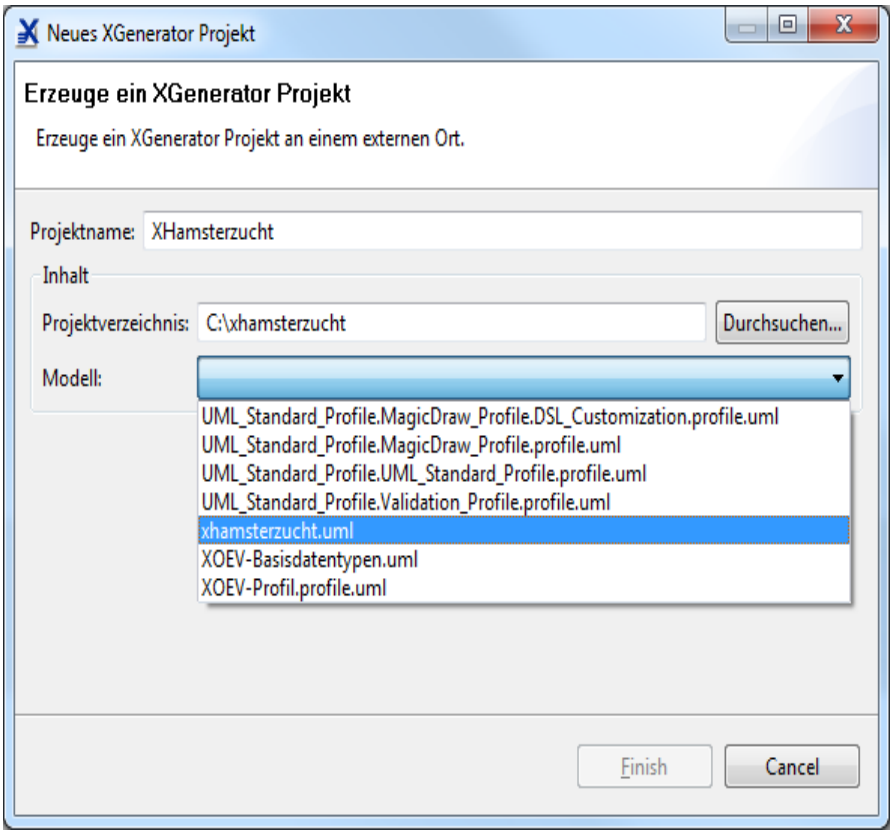
Ein neues Projekt kann erstellt werden, wenn es zuvor wie in Abschnitt 9.2.1.3 auf Seite 137 beschrieben, vorbereitet wurde. Bei Klick auf den Button "Erstellen" öffnet sich ein weiterer Dialog.



Unter "Projektname" kann ein frei wählbarer Name (z.B. "XHamsterzucht") eingetragen werden, der dann in der Oberfläche angezeigt wird. Als Projektverzeichnis wird das wie in Abschnitt 9.2.1.3 auf Seite 137 eingerichtete Projektverzeichnis ausgewählt. Ein Projektverzeichnis kann nur für ein Projekt verwendet werden.

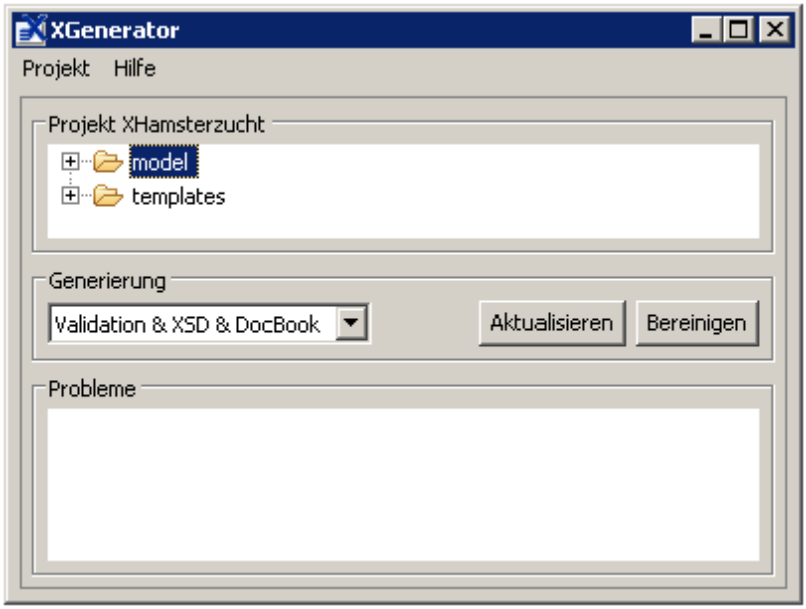


Nach Auswahl des Projektverzeichnisses werden unter Modell die XMI-Dateien (*.uml) des model-Ordners angezeigt. Diese beziehen sich, wie in Abschnitt 9.2.1.3.1 auf Seite 138 erläutert, auf das UML-Modell und die Profile. Existiert in dem ausgewählten Projektverzeichnis kein Unterordner "model", so bleibt die Auswahl unter "Modell" inaktiv.



Aus den unter "Modell" angezeigten Dateien wird das eigentliche Fachmodell ausgewählt.

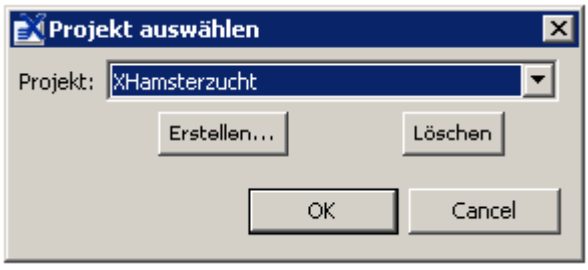
Nach Betätigen des Buttons "Finish" ist das Projekt erstellt und kann mit "OK" geöffnet werden.

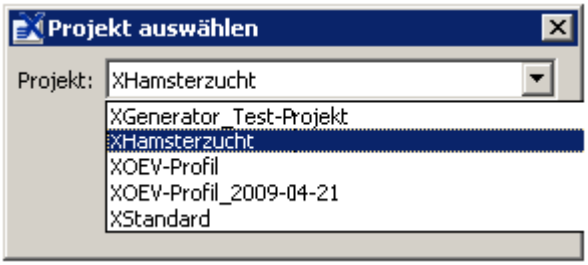


In der Oberfläche des XGenerators wird das vorbereitete Projekt mit Namen und als Dateibaum mit allen vorbereiteten Unterverzeichnissen des Projektordners angezeigt. In diesem Dateibaum kann navigiert und die ausgewählten Dateien mit ihren zu den Endungen gehörigen Programmen geöffnet werden.

9.2.2.4 Projekt öffnen und wechseln

Bereits einmal erstellte Projekte können über den Dialog unter dem Menüpunkt "Verwalten" direkt wieder geöffnet bzw. auch gewechselt werden, wenn bereits mehrere Projekte eingerichtet wurden.

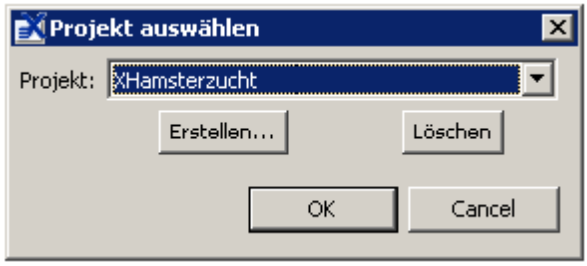




Nach Klick auf "OK" öffnet sich das ausgewählte Projekt im XGenerator.

9.2.2.5 Projekt löschen

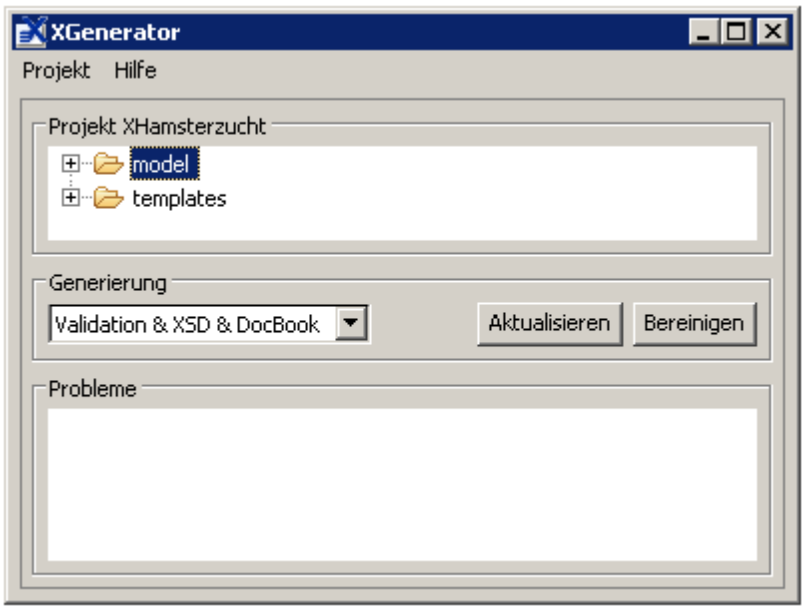
Über denselben Verwaltungsdialog wie im vorherigen Abschnitt können ausgewählte Projekte mit einem Klick auf "Löschen" entfernt werden.



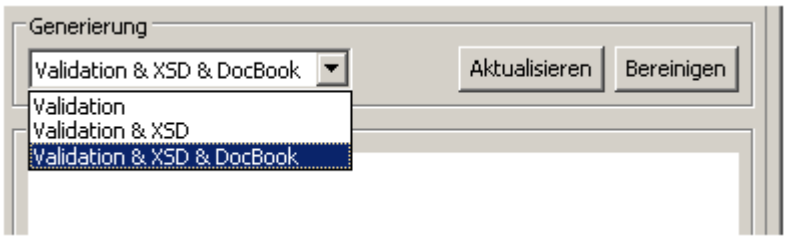
Hinweis: Ein Projekt zu löschen bedeutet, es aus dem Workspace, d.h. aus der "Anzeige" des XGenerators zu löschen. Physisch bleiben die Dateien im Dateisystem erhalten.

9.2.3 Automatisierte Erstellung von XML-Schema-Dateien und DocBook-Fragmenten

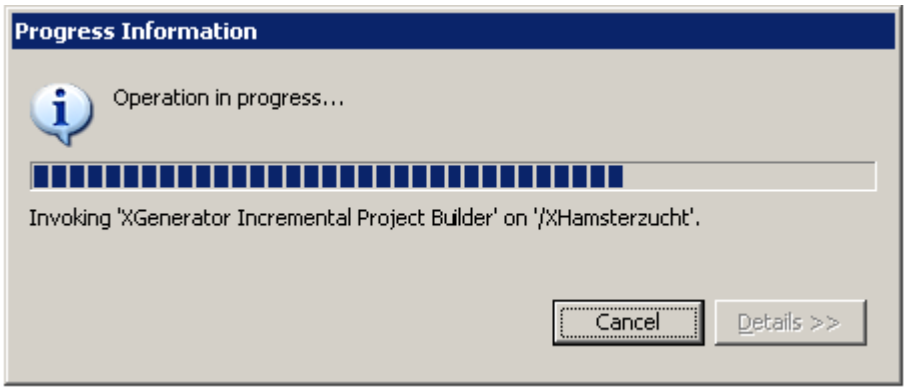
Nachdem ein Projekt im XGenerator geöffnet wurde,



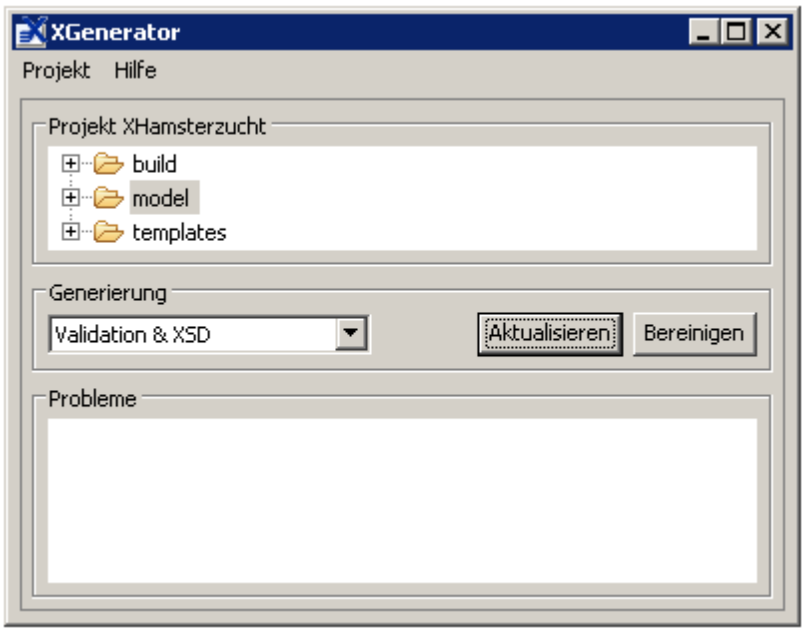
kann das UML-Modell gegen das angewandte Profil validiert werden bzw. können die Schema- bzw. DocBook-Dateien generiert werden.



Welche dieser Arbeitsschritte vom Tool ausgeführt werden sollen, wird über eine Auswahlbox gesteuert. Nach Auswahl eines Modus, z. B. "Validation & XSD'", wird nach Klick auf den Button "Aktualisieren" der Validierungs- und Generierungsprozess für XML-Schema-Dateien angestoßen. Der Prozess wird sequentiell gemäß der Vorgabe in der Auswahlbox abgearbeitet.



Der angestoßene Prozess kann über "Abbrechen" auf dem Prozessfortschrittsbalken beendet werden. Der Abbruch der laufenden Operation kann jedoch einige Zeit dauern, da das Tool Aufgaben in größeren Blöcken abarbeitet und nur vor Beginn der nächsten Teilaufgabe die Benutzereingaben auswertet, um gegebenenfalls abzubrechen.



Wenn der Unterordner "build" nicht manuell angelegt wurde, wird dieser beim ersten Bearbeiten eines Projektes (gestartet mit dem Button "Aktualisieren") im Projektverzeichnis erzeugt. Die vom Tool generierten Dateien werden in die entsprechend festgelegten Unterordner geschrieben. Zusätzlich wird ein aktuelles Protokoll "generierung.log" in den Build-Ordner geschrieben (Details siehe Abschnitt 9.2.3.5 auf Seite 151).

Mit dem Button "Bereinigen" werden die erzeugten Dateien auch physisch aus dem Dateisystem entfernt.

Im angezeigten Dateibaum kann navigiert und die Dateien wie Vorlagen, Schema-Dateien usw. können aus dem Dateibaum des XGenerators heraus mit dem der jeweiligen Dateiendung zugeordneten Programm geöffnet und bearbeitet werden.

Im folgenden werden die einzelnen Arbeitsschritte mit dem XGenerator zur Erzeugung der XML-Schema-Dateien und DocBook-Fragmente näher beschrieben.

9.2.3.1 Einlesen und Validieren des Modells

Neben dem eigentlichen UML-Modell und dem angewandten Profil werden auch die weiteren, im Verzeichnis "model" enthaltenen Dateien (siehe Abschnitt 9.2.1.3.1 auf Seite 138) vom XGeneratoreingelesen.

Nach dem Einlesen der Dateien erfolgt die Validierung des UML-Modells z. B. auf die korrekte Anwendung des XÖV-Profils. Hierzu werden u. a. auch die zuvor aus "model" eingelesenen invariants- und operations-Dateien benötigt (siehe Abschnitt 9.1.2 auf Seite 133). Fehler z. B. bei der Anwendung des Profils auf das UML-Modell werden durch die Validierung aufgedeckt und vom XGeneratorin textueller Form gemeldet (siehe Abschnitt 9.2.3.6 auf Seite 152). Die Meldungen sollen den Anwender befähigen, die Fehler aufzufinden und zu korrigieren.

9.2.3.2 Generieren und Validieren der XML-Schema-Dateien

Auf Basis einer erfolgreichen Validierung gegen ein Profil, kann die von den XÖV-XSD-Vorlagen gesteuerte Generierung der XML-Schema-Dateien (*.xsd) erfolgen. Die Vorlagen arbeiten mit den OCL-Ausdrücken der ebenfalls aus dem Model-Ordner eingelesenen operations-Dateien (siehe Abschnitt 9.1.2 auf Seite 133). Die Ergebnisse werden in das Projektverzeichnis (z. B. "XHamsterzucht\build\xsd") geschrieben.

Nach der XML-Schema-Generierung wird geprüft, ob die erzeugten XML-Schema-Dateien valide gemäß W3C sind. Die Prüfung erfolgt auf Basis einer EMF (Eclipse Modelling Framework)-internen Schema-Validierung. Fehler, die bei der Generierung bzw. Validierung festgestellt werden, meldet der XGenerator ebenfalls.

9.2.3.3 Generierung von DocBook-Dateien

Der XGenerator bietet zur weiteren Verarbeitung des UML-Modells die Möglichkeit, über die Auswahl von "Validation & XSD & DocBook" weitere Dateien zu erzeugen.

Nach erfolgreicher Schema-Generierung und -Validierung können, gesteuert über z. B. die XÖV-DocBook-Mustervorlagen, DocBook-Dateien (*.xml) erzeugt werden. Die vom XGenerator erzeugten DocBook-Fragmente sind xml-Dateien, die die zuvor erzeugten XML-Schema-Dateien auf Basis des UML-Modells als auch auf Basis der Schema-Dateien selbst dokumentieren.

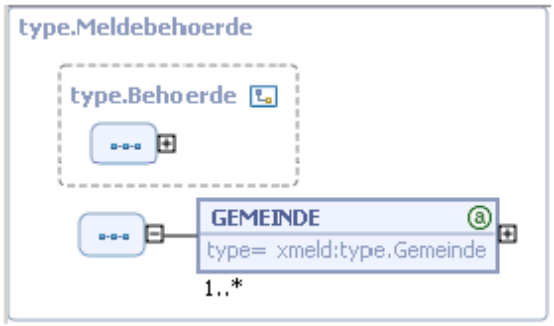
Hinweis: Die XÖV-DocBook-Mustervorlagen sind im Zuge der XÖV-Konformität optional anzuwenden. Sie können spezifisch durch ein einzelnes XÖV-Vorhaben auf die eigenen Bedürfnisse angepasst bzw. durch eigene ersetzt werden – es besteht auch die Möglichkeit, die Dokumentation auf einem anderen Weg zu erzeugen, z. B. durch die direkte Generierung aus dem UML-Case-Tool.

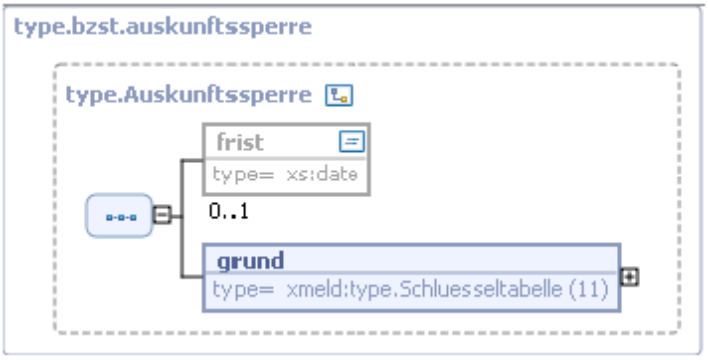
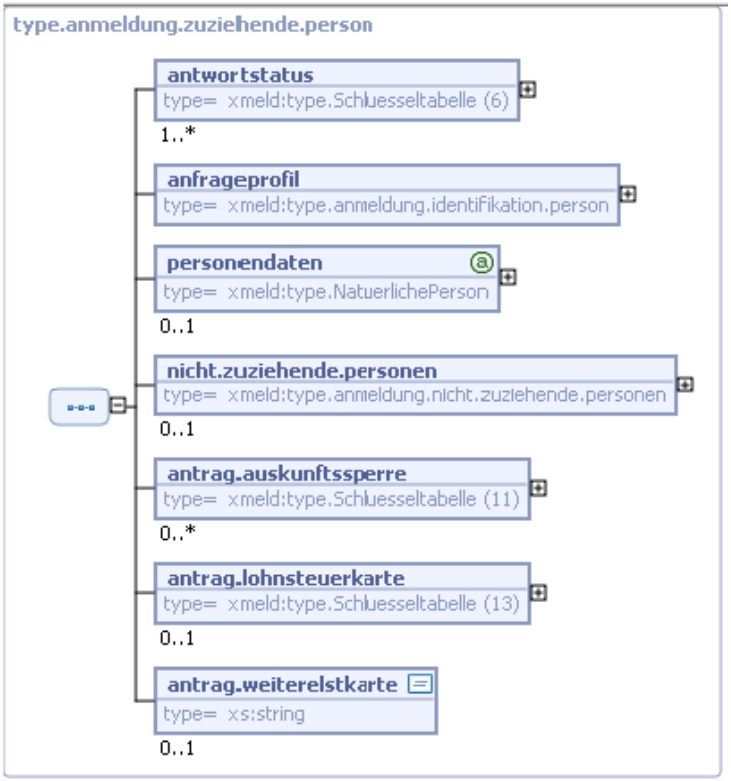
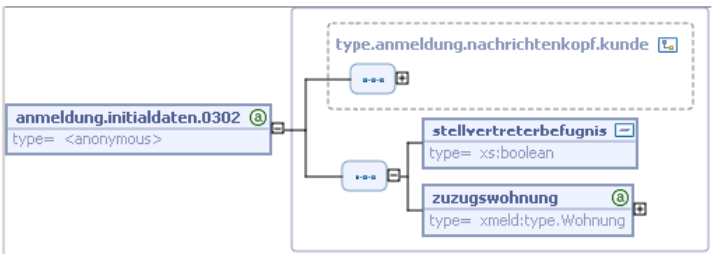
Die Ergebnisse werden in das Projektverzeichnis (z. B. "XHamsterzucht\build\docbook") geschrieben. Die Struktur des "docBook"-Verzeichnisses mit seinen Unterverzeichnissen ergibt sich aus den Festlegungen in den DocBook-Vorlagen. Die erzeugten DocBook-Dateien sind Teil der Dokumentation, welche in der Regel aus weiteren, auf anderen Wegen, z. B. manuell erstellten DocBook-Dateien besteht. Bei der Generierung auftretende Fehler, z. B. durch unkorrekte Anweisungen in den Vorlagen, werden vom XGenerator angezeigt.

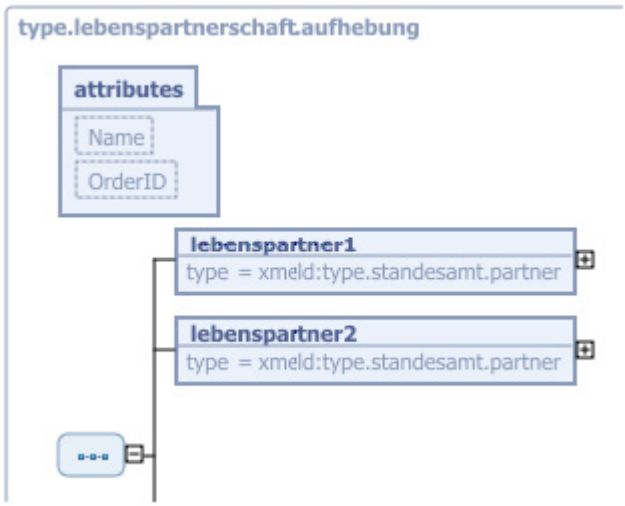
In diesem Arbeitsschritt der DocBook-Generierung werden auch SVG-Grafiken (*.svg) zur grafischen Dokumentation von XML-Schema-Elementen erzeugt, die auf die bereits zuvor generierten XML-Schema-Dateien zugreifen.

Die SVG-Grafiken werden durch einen XGenerator-eigenen Prozessor generiert, der durch die DocBook-Vorlagen angestoßen wird, und haben ein fest vorgegebenes Layout. Die im Folgenden aufgeführten Erläuterungen zu den verwendeten Icons bzw. Beispielen von Grafiken sollen helfen, die allgemeine Darstellung der Schema-Grafiken eindeutig zu interpretieren.

Icon	Bedeutung
	zeigt an, dass eine Attributdefinition vorliegt (ohne Icon sind Elementdefinitionen dargestellt)
	zeigt an, dass das Element einen W3C-Basistypen als Datentyp besitzt
	zeigt an, dass der Typ ein komplexer Typ ist
	zeigt an, dass eine Sequenz von Elementen und Typen definiert ist
	zeigt an, dass eine Auswahl von Elementen definiert ist

Beispiele für Grafiken	Bedeutung
	Darstellung eines benannten complex-Type: Hier wird deutlich, dass "frist" ein simpleType und "Grund" vom Typ Schlüsseltabelle ist.
	Darstellung einer Extension: Das Beispiel zeigt, dass type.Meldebehoerde eine Extension von type.Behoerde ist.

Beispiele für Grafiken	Bedeutung
	Darstellung einer Restriction: Das Beispiel zeigt, dass type.bzst.auskunftssperre eine Restriktion von type.Auskunftssperre ist und nur aus dem Element "grund" besteht
	Darstellung von Kardinalitäten: An dieser Grafik wird die Darstellung der Kardinalität von Elementen deutlich. Elemente ohne Angaben haben genau die Kardinalität "1". Eine abweichende Kardinalität wird separat angegeben, z.B. "1..*", "0..1" oder "0..*".
	Darstellung eines anonymen Typen: Ein anonymer complexType wird aufgeklappt dargestellt, damit seine Elemente erkennbar sind.

Beispiele für Grafiken	Bedeutung
	Darstellung eines Attributs: Ein Attribut wird gruppiert mit ggf. anderen Attributen innerhalb eines Typs dargestellt.

9.2.3.4 Validieren der DocBook-Dateien

Nach Erzeugung der DocBook-Dateien findet deren Validierung statt. Diese inkludiert auch manuell erzeugte DocBook-Dateien. Die Validierung erfolgt auf der Basis der `spezifikation.xml` im Verzeichnis ../src, die auch den Verweis auf die der Validierung zugrundeliegende DocBook-DTD enthält. Mit der DocBook-Validierung über alle Dateien wird z. B. sichergestellt, dass sämtliche Referenzen über Pfadangaben innerhalb der Dateien korrekt sind. Auftretende Fehler werden auch in diesem Fall vom XGenerator angezeigt.

Nach erfolgreicher Validierung können diese DocBook-Dateien in einem, der Verwendung des XGenerators nachgeordneten und für jedes XÖV-Vorhaben spezifischen Prozess für die Erstellung der Dokumentation z. B. in Form von HTML oder PDF genutzt werden.

9.2.3.5 Protokollierung

Neben der Anzeige eines Prozessfortschrittsbalkens wird der gesamte Prozessverlauf der Validierung und Generierung vom XGenerator in der Datei `build/generierung.log` protokolliert und kann z. B. über das Öffnen mit einem Text-Editor synchron verfolgt werden. Bei einem erneuten Prozessdurchlauf wird der bestehende Inhalt der Log-Datei mit den aktuellen Ergebnissen überschrieben, d. h. nur der jeweils letzte Lauf eines Projektes ist hier protokolliert.

Die Log-Datei hat einen dem Prozess entsprechenden Aufbau mit folgendem Inhalt:

Inhalt	Bedeutung
Generiere Projekt 'XHamsterzucht'	Name des Projektes
Projektordner: C:\.\xhamsterzucht	Pfadangabe des Projektordners
Startzeit:	Datum und Startzeit der Validierung und Generierung
Teilaufgabe 'model'	Angaben zur UML-Profilvalidierung des Modells
Teilaufgabe 'xsdGenerator'	Auflistung der Namen der generierten XML-Schema-Dateien

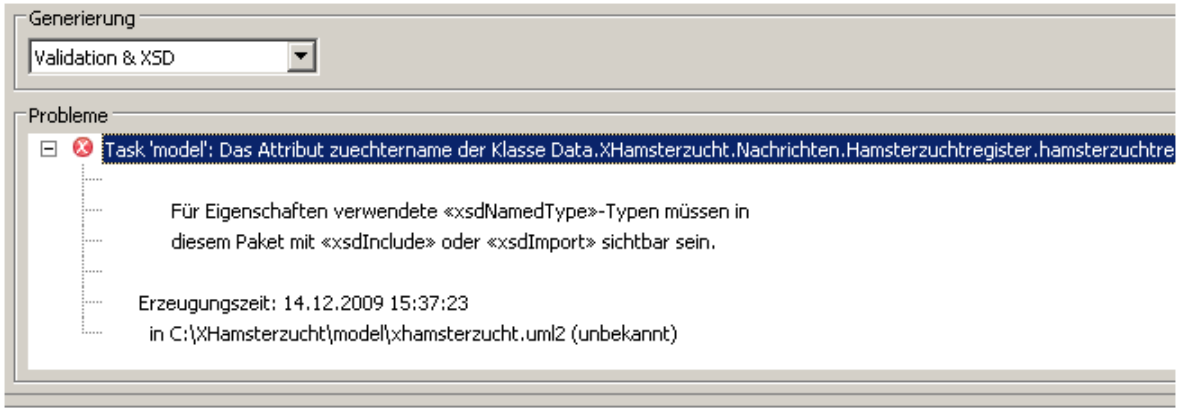
Inhalt	Bedeutung
Teilaufgabe 'xsd'	Angaben zur Schema-Validierung
Teilaufgabe 'docBookGenerator'	Auflistung der Namen der generierten DocBook-Dateien
Teilaufgabe 'xml'	Angaben zur DocBook-Validierung
Endzeit:	Datum und Endzeit der Validierung und Generierung

Treten im Verlauf des Prozesses Fehler auf, wird im entsprechenden Teilprozess (Teilaufgabe) und am Ende des Protokolls allgemein darauf hingewiesen.

Im Verzeichnis ./xgenerator des Produktverzeichnisses (Windows- oder Linux-Version) gibt es eine Datei xgenerator0.log, die technisch detaillierter auflistet, welche Tasks/Teilaufgaben bei einem Durchlauf des XGenerators abgearbeitet wurden bzw. welche Fehler aufgetreten sind. Diese Datei wird fortlaufend weitergeschrieben und kann für Zwecke der Fehleranalyse genutzt werden. Im Falle, dass diese Datei zu viel Speicherplatz beansprucht, kann sie problemlos manuell gelöscht werden. Bei einem weiteren Durchlauf wird diese Datei neu generiert.

9.2.3.6 Fehlermeldungen

Kommt es im Prozess des Einlesens, Validierens oder Generierens zu Fehlern, wird der Prozess an dieser Stelle abgebrochen und detaillierte textuelle Fehlermeldungen mit Angaben zur Datei, in der der Fehler auftritt, werden in dem separaten Problemfenster des XGenerators ausgegeben.

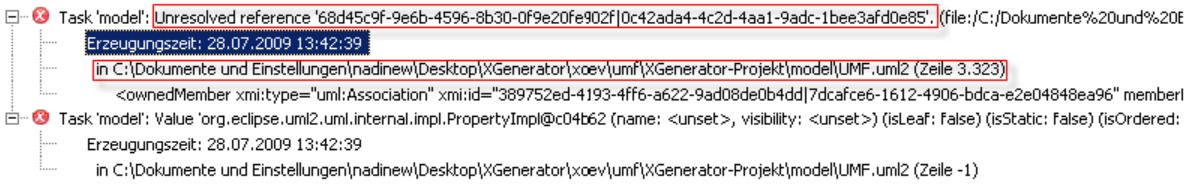


Nach dem Korrigieren der Fehler, z.B. durch Bereitstellen eines korrigierten Modells oder Änderungen an Vorlagen, kann mit dem Button "Aktualisieren" der Prozess fortgesetzt bzw. neu gestartet werden.

Eine Auswahl an Fehlermeldungen, die aus einem unsachgemäßen Umgang mit dem XGenerator zurückzuführen sind, ist nachfolgend zu finden.

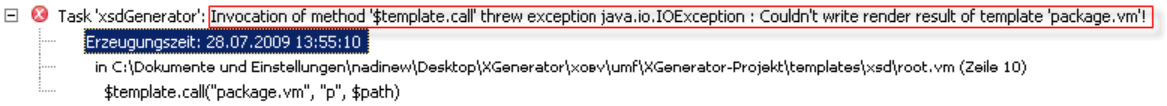
9.2.3.6.1 Beispiel: Fehlerhafte Referenzierungen zwischen und in XMI-Export-Dateien

Fehlermeldungen mit Angabe einer "unresolved reference" auf eine bestimmte ID sowie dem Verweis auf die XMI-Datei (hier: UMF.uml2) haben ihre Ursache in einer fehlerhaften Überführung des UML-Modells in das XMI. Die Generierung der XML-Schema-Dateien wird abgebrochen.



9.2.3.6.2 Beispiel: XML-Schema in anderem Programm geöffnet

Fehlermeldungen, die eine Java Exception beinhalten, verweisen darauf, dass der XGenerator nicht ordnungsgemäß arbeiten kann. Das Beispiel zeigt eine typische Fehlermeldung java.io.IOException, die ihre Ursache in einem generierten, geöffneten XML-Schema hat. Der XGenerator kann daher den Ordner xsd im Verzeichnis build nicht löschen und generiert keine XML-Schema-Dateien. Das Schließen des geöffneten XML-Schemas (manchmal auch erst das Beenden des Programms, mit dem die Datei geöffnet wurde) beseitigt den Fehler.



A Anhang zum Basisdatentyp String.Latin

Entfallen.

Der Datentyp String.Latin ist Bestandteil des eigenständigen Standards Lateinische Zeichen in Unicode (verfügbar unter http://xoev.de/latinchars/).

B Überblick zu Velocity und OCL

Die Invarianten (z.B. XÖV-Invarianten), Templates (z.B. XÖV-XSD-Vorlagen) und Operationen (z.B. XÖV-XSD-Operationen) beinhalten bezüglich ihres Aufbaus Aspekte aus zwei verschiedenen Bereichen, die in den nachfolgenden Abschnitten kurz beschrieben werden sollen:

1. Velocity-Syntax für die Steuerung des Ablaufs in den Templates
2. OCL-Abfragen auf Inhalte im UML-Modell für Templates, Invarianten und Operations

B.1 Velocity

Grundlegend wird festgestellt, dass die Templates mittels Velocity[1] so aufgebaut sind, dass alles, was keine spezielle Anweisung darstellt, durch den im XGenerator 2 integrierten Template-Prozessor, direkt ausgegeben wird. So wird in dem untenstehenden Beispiel mit "Hallo" eine Ausgabe getätigt und mit "#" eine Anweisung eingeleitet.

```
Hallo
#set($status="gegruesst")
```

Innerhalb der Templates werden die folgenden Velocity-Anweisungen verwendet:

Anweisungstyp	Beschreibung
`#set($var="value")`	Setzen eines Wertes für eine Variable

`#if(condition) output`
`  (optional: #else output oder #elseif(condition) output)`
`#end`
	Ausführen einer bedingten Auswertung, wenn die Bedingung mit "true" erfüllt ist (es kann eine beliebige Schachtelung von bedingten Auswertungen vorgenommen werden)

`#foreach($var in value)`
`  output`
`#end`
	Ausführen einer Schleife über eine Menge von Elementen, wobei der Output für jedes Element einmal ausgegeben wird (es kann eine beliebige Schachtelung von Schleifen vorgenommen werden)

`$var`	Auslesen eines Variablenwertes
`$template.call(templatename, parameters [,target])`	Aufruf eines anderen Templates und Übergabe der dazugehörigen Parameter (optional kann auch eine neue Ausgabedatei für das aufgerufene Template angegeben werden)

1. http://velocity.apache.org

B.2 Object Constraint Language (OCL)

Mittels der Object Constraint Language (OCL)[1] kann auf die Elemente des UML-Modells zu einem XÖV-Standard, das im Format EMF UML2 (v1.x) XMI vorliegt, zugegriffen werden sowie eine Auswertung dieser erfolgen. Die OCL ist eine ergänzende Spezifikation zu UML.

Der vom XGenerator verwendete OCL-Auswerter wird über die Variable "ocl" in den Dateien `global_variables.vm` zur Verfügung gestellt. Durch diese Variable wird eine Methode "eval" zur Verfügung gestellt, an die OCL-Abfragen in den Templates gestellt werden können und entsprechende Ergebnisse zurückgeliefert werden – in Invarianten ist die Angabe der Methode nicht erforderlich.

Velocity-Variablen können in OCL-Abfragen ebenfalls verwendet werden – sie müssen jedoch dann in Abweichung zur sonstigen Velocity-Syntax ohne ein Dollarzeichen angegeben werden.

```
#foreach($paket in $ocl.eval("Package.allInstances"))
  - $ocl.eval("paket.name")
#end
```

Darüber hinaus lassen sich komplexe OCL-Abfragen mittels Hilfsoperationen zu den UML-Metaklassen strukturieren und systematisieren. Ein weiterer Nebeneffekt ist die Wiederverwendbarkeit von Abfragen durch die Verwendung von Hilfsoperationen.

Eine Hilfsoperation wird immer mit den folgenden Eigenschaften beschrieben:

- Kontext: auf welche UML-Metaklasse (Package, Type, Class, Classifier, etc.) bezieht sich die Operation
- Name: mit welchem Namen kann die Operation in OCL-Abfragen aufgerufen werden
- Parameter: wie heißt der übergebene Parameter und von welchem Typ ist er (sofern ein Parameter übergeben werden muss)
- Inhalt: welche OCL-Abfragen sollen strukturiert werden
- Dokumentation: welchen Zweck erfüllt die Operation

Nachfolgend ist ein Beispiel zu einer Operationsdefinition abgebildet.

```
<operation context='Class' name='hlpDocBookTitle'>
  <result type='String'/>
  <body>
    Sequence{self.extension_xsdTitled.title}-&gt;iterate(title;className:String=self.name|
      if title.isDefined and title&lt;&gt; '' then title else className endif)
  </body>
  <documentation>Falls diese Klasse das 'title'-Attribut gesetzt hat (möglich bei xsdNamedType,xsdMessage) wird dieser Wert geliefert, andernfalls der Name der Klasse.</documentation>
</operation>
```

Der entsprechende Aufruf der Operation wäre dann wie folgt abgebildet:

```
_
<row>
  <entry spanname="all">Element: $ocl.eval("e.hlpDocBookTitle()")</entry>
</row>
_
```

1.http://www.omg.org/technology/documents/modeling_spec_catalog.htm#OCL

C XÖV-Glossar

Dieses Glossar umfasst Begriffe, die für die Arbeit im XÖV-Umfeld gedacht sind. Einige allgemeine Begriffe sind daher auf ihre Verwendung im XÖV-Kontext eingeschränkt und nicht allumfassend definiert.

Aktivitätsdiagramm

Ein Aktivitätsdiagramm ist eine Diagrammart der UML. Es beschreibt einen Ablauf und wird durch verschiedene Arten von Aktionen definiert, die durch Objekt- und Kontrollflüsse miteinander verbunden sind. Ein Aktivitätsdiagramm befasst sich mit den Verhaltensabläufen eines Systems.

⇨Unified Modeling Language

Anwendungsfalldiagramm

Ein Anwendungsfalldiagramm ist eine Diagrammart der UML und beschreibt die Anforderungen an ein System in Form seiner Anwendungsfälle und den daran beteiligten Akteuren bzw. Systemen. Anwendungsfalldiagramme beschreiben kein Verhalten und keine Abläufe, sondern die Zusammenhänge der an den Anwendungsfällen beteiligten Systeme und Akteure.

⇨Unified Modeling Language

Codeliste

Eine Codeliste ist eine Liste von Werten, die den Wertebereich eines zu übermittelnden Datenfeldes einschränkt.

Datenmodell

Ein Datenmodell beschreibt Strukturen und Beziehungen von Daten. Die Beschreibung des Modells erfolgt durch eine formale Darstellung beispielsweise durch ein UML-Klassendiagramm.

⇨Unified Modeling Language

⇨Klassendiagramm

DocBook

DocBook ist ein präsentationsneutrales Format für technische Artikel, Bücher und Referenzdokumente. DocBook ist eine Auszeichnungssprache, basiert u.a. auf XML und wird als offener Standard von OASIS gepflegt. Für DocBook existieren eine Reihe frei verfügbarer Werkzeuge, mit deren Hilfe HTML, PDF und diverse andere Ausgabeformate erzeugt werden können

Link: http://www.oasis-open.org/docbook/

⇨eXtensible Markup Language

⇨Organization for the Advancement of Structured Information

eXtensible Markup Language

Die eXtensible Markup Language (XML) ist eine Formatbeschreibungssprache für den Austausch strukturierter Daten und wurde 1997 vom W3C standardisiert.

Link: http://www.w3c.org/XML/

⇨World Wide Web Consortium

Fachexperte

Person, die die fachlichen Anforderungen an einen Standard definiert, ohne auf die technische Ausgestaltung gemäß XÖV einzugehen.

⇨Standard

⇨XÖV

Fachmodell

Bezeichnet ein Modell einer konkreten Fachlichkeit, welches z.B. als UML-Modell abgebildet das Datenmodell beinhaltet.

Fachverfahren

Für eine spezifische Fachaufgabe entwickelte IT-Lösung.

Genericode

XML-Format zur Beschreibung von Codelisten, das von der OASIS veröffentlicht wurde.

Link: http://docs.oasis-open.org/codelist/ns/genericode/1.0/

⇨Codeliste

⇨eXtensible Markup Language

⇨Organization for the Advancement of Structured Information

Genericoder

Der Genericoder ist ein von der XÖV-Koordination bereitgestelltes XÖV-Werkzeug. Er transformiert als CSV-Textdateien vorliegende Codelisten in eine XML-Datei im Format OASIS Genericode Version 1.0.

Link: https://joinup.ec.europa.eu/software/genericoder/home

⇨Codeliste

⇨eXtensible Markup Language

⇨Organization for the Advancement of Structured Information

⇨Genericode

⇨XÖV-Koordination

⇨XÖV-Werkzeug

Invariante

Invarianten sind Regeln, die Teil eines UML-Profils sind, und überprüfen mittels OCL-Abfragen z.B. die korrekte Verwendung von Stereotypen.

⇨OCL

⇨Operation

⇨Stereotyp

⇨UML-Modell

Klassendiagramm

Ein Klassendiagramm ist eine Diagrammart der UML, die Klassen von Datenobjekten eines Systems und ihre Beziehungen untereinander darstellt. Zu einer Klasse werden weitere Informationen z. B. in Form von Attributen angegeben.

⇨Unified Modeling Language

OASIS

⇨Organization for the Advancement of Structured Information

Object Constraint Language

Die Object Constraint Language (OCL) ist ein Bestandteil der UML, der von der OMG herausgegeben wird, zur Definition von u.a. Invarianten sowie Abfragen in Operationen und Templates bezüglich der Bestandteile eines UML-Modells.

Link: http://www.omg.org/technology/documents/modeling_spec_catalog.htm#OCL

⇨Invariante

⇨Object Management Group

⇨Operation

⇨Template

⇨Unified Modeling Language

Object Management Group

Die Object Management Group (OMG) ist ein Konsortium, das sich mit der Entwicklung von Standards für die herstellerunabhängige systemübergreifende objektorientierte Programmierung beschäftigt.

Link: http://www.omg.org

OCL

⇨Object Constraint Language

OMG

⇨Object Management Group

Online Services Computer Interface

Online Services Computer Interface (OSCI) ist ein Branchenstandard der öffentlichen Verwaltung für die sichere Transaktion von Geschäftsprozessen über offene Netze, wie bspw. das Internet.

Link: http://www.osci.de

Operation

Eine häufig genutzten OCL-Abfrage, die im Rahmen von Invarianten und Templates wiederverwendet wird.

⇨Invariante

⇨Object Constraint Language

⇨Template

Organization for the Advancement of Structured Information

Die Organization for the Advancement of Structured Information Standards (OASIS) ist eine internationale, nicht gewinnorientierte Organisation, die sich mit der Entwicklung von E-Business- und Web-Service-Standards beschäftigt.

Link: http://www.oasis-open.org

OSCI

⇨Online Services Computer Interface

Produktionsumgebung

Eine Produktionsumgebung adaptiert die Bestandteile des XÖV-Produktionszubehörs an die Anforderungen an ein bzw. eine Reihe von XÖV-Vorhaben. So wird ein spezifisches UML-Werkzeug inkl. der jeweiligen Implementierungen des XÖV-UML-Profils, der XÖV-Basisdatentypen und der XÖV-Kernkomponenten bereitgestellt. Weiterhin kann der XGenerator um spezifische Invarianten zur UML-Modell-Validierung sowie spezifische DocBook-Vorlagen (Templates) zur Dokumentationserzeugung ergänzt werden. Auch die Anreicherung der Software-Werkzeuge um eine Anwendung zur Dokumentationsgenerierung aus den Fragmenten ist in einer Produktionsumgebung möglich. Produktionsumgebungen werden nicht durch die XÖV-Koordination bereitgestellt und gepflegt, sondern durch das jeweilige XÖV-Vorhaben selbst.

⇨DocBook

⇨Invariante

⇨Template

⇨UML-Modell

⇨UML-Modellierungswerkzeug

⇨XÖV-Basisdatentyp

⇨XÖV-Kernkomponente

⇨XÖV-Koordination

⇨XÖV-Produktionszubehoer

⇨XÖV-UML-Profil

⇨XÖV-Vorhaben

Scalable Vector Graphics

Scalable Vector Graphics ist eine vom W3C empfohlene Spezifikation zur Beschreibung zweidimensionaler Vektorgrafiken basierend auf XML.

Link: http://www.w3.org/Graphics/SVG/

⇨eXtensible Markup Language

⇨World Wide Web Consortium

Semantik

Die Semantik definiert – im Gegensatz zur Syntax – die Bedeutung der gültigen Zeichen, Wörter und Sätze einer Sprache. So ist die Dokumentation eines XML-Schema-Elements in einer XML-Schema-Datei ein Beispiel für die Festlegung der Semantik eines Datenfeldes, d.h. wie es zu verstehen ist.

⇨Syntax

⇨XML-Schema

Standard

Ein Standard im Sinne dieses Dokumentes ist ein abgestimmtes Fachmodell für ein Datenaustauschformat. Er beinhaltet eine einheitliche und eindeutige Definition und Beschreibung von Syntax und Semantik von Datenstrukturen und Nachrichten für die elektronische Datenübermittlung innerhalb und mit der öffentlichen Verwaltung.

⇨Fachmodell

⇨Semantik

⇨Syntax

Stereotyp

Ein Stereotyp erlaubt es, zu einem Bestandteil eines UML-Modells weitere Klassifikationen und Eigenschaften zuzuordnen, z.B. zur weiteren technischen Spezifizierung. Ein Stereotyp wird als Bestandteil eines UML-Profils definiert.

⇨UML-Modell

⇨UML-Modell

SVG

⇨Scalable Vector Graphics

Syntax

Die Syntax definiert, wie gültige Sätze einer Sprache aufgebaut werden. So besteht eine Sprache aus einer Menge gültiger Symbole (Zeichen, Wörter) und einem Regelwerk (Grammatik), das besagt, wie die einzelne Zeichen oder Wörter miteinander kombiniert werden, um gültige Sätze zu formen. Die Syntax trifft aber keine Aussage über die Bedeutung (Semantik) der gebildeten Sätze. So ist beispielsweise für eine XML-Schema-Datei die Syntax über die XML Schema Definition des W3C definiert, die Vorgaben zu gültigen Zeichen und dem grundsätzlichen Aufbau von XML-Schema-Dateien macht.

⇨Semantik

⇨World Wide Web Consortium

⇨XML Schema Definition

⇨XML-Schema

Template

Ein Template wird in der Sprache Velocity erstellt und enthält Abläufe zur Steuerung von Datei-Generierungen und OCL-Aufrufe zur Navigation und Auswertung im UML-Modell.

⇨DocBook

⇨Object Constraint Language

⇨UML-Modell

⇨Velocity

⇨XGenerator

⇨XML-Schema

⇨XÖV-XSD-Vorlagen

⇨XÖV-DocBook-Mustervorlagen

UML-Modell

Sprachelement der UML, das Modellelemente für die Beschreibung der Struktur und des Verhaltens eines Systems abbildet. Die Abbildung auf die unterschiedlichen Sichten des Systems erfolgt über unterschiedliche UML-Diagramme.

⇨Aktivitätsdiagramm

⇨Anwendungsfalldiagramm

⇨Klassendiagramm

⇨Unified Modeling Language

UML-Modellierungswerkzeug

Werkzeug zur Erstellung von UML-Modellen.

⇨UML-Modell

UML-Modell

UML-Profile sind ein Standardmechanismus zur Erweiterung oder Anpassung des Sprachumfangs der UML an fachliche oder technische Anforderungen und enthalten Sprachkonzepte, die über UML-Basiskonstrukte festgelegt werden. Betrachtet man UML als “Sprachfamilie”, dann sind UML-Profile Elemente – also konkrete Sprachen – dieser Familie.

⇨Unified Modeling Language

UN/CEFACT

⇨United Nations Center for Trade Facilitation and Electronic Business

Unified Modeling Language

Die Unified Modeling Language (UML) ist eine grafische Modellierungssprache für den Entwurf und die Entwicklung von Software-Systemen. Sie wird durch die OMG veröffentlicht.

Link: http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML

⇨Object Management Group

United Nations Center for Trade Facilitation and Electronic Business

Die United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) ist eine internationale Standardisierungsorganisation für die Abstimmung einheitlicher Fachdatenschnittstellen für das E-Business und das E-Government. Die Entwicklung erfolgt für alle Branchen und Länder, wodurch eine weltweite Verwendung angestrebt wird. Die Methoden und Modelle hängen nicht von bestimmten Technologien oder Implementierungen ab. Dennoch können die Modelle auf eine standardisierte Weise in die derzeitigen Technologien, wie XML und XML-Schema abgebildet werden.

Link: http://www.unece.org/cefact

⇨eXtensible Markup Language

⇨XML-Schema

Velocity

Sprache zur Definition von Abläufen und Ausgaben in Form von Templates.

Link: http://velocity.apache.org

⇨Template

W3C

⇨World Wide Web Consortium

Web Services Description Language

Die Web Service Description Language (WSDL) ist eine vom W3C veröffentlichte Platform-, Programmiersprachen- und Protokoll-unabhängige Beschreibungssprache für Netzwerkdienste (Web Services) zum Austausch von Nachrichten auf Basis von XML. WSDL ist eine Metasprache, mit deren Hilfe Funktionen, Daten, Datentypen und Datenaustauschprotokolle eines Netzwerkdienstes beschrieben werden können.

Link: http://www.w3.org/TR/wsdl20/

⇨eXtensible Markup Language

⇨World Wide Web Consortium

World Wide Web Consortium

Das World Wide Web Consortum (W3C) ist ein internationales Gremium, das sich mit der Erstellung von Technologien für das World Wide Web befasst.

Link: http://www.w3c.org

WSDL

⇨Web Services Description Language

XGenerator

Mit dem durch die XÖV-Koordination bereitgestellten XÖV-Werkzeug XGenerator können aus einem UML-Modell mittels Templates XML-Dateien erzeugt werden. Für die konkrete Verwendung im XÖV-Kontext werden XÖV-XSD-Vorlagen (Templates), XÖV-XSD-Operationen, XÖV-DocBook-Mustervorlagen (Templates) und XÖV-DocBook-Musteroperationen verwendet, um automatisiert XML-Schemata und eine dazu konsistente Dokumentation in DocBook-XML-Dateien mit SVG-Grafiken zu generieren. Voraussetzung für den XÖV-konformen Einsatz des XGenerators ist die Erstellung eines UML-Modells unter Anwendung des XÖV-UML-Profils.

⇨DocBook

⇨Scalable Vector Graphics

⇨Template

⇨UML-Modell

⇨Unified Modeling Language

⇨XML-Schema

⇨XÖV-Konformität

⇨XÖV-Koordination

⇨XÖV-DocBook-Musteroperationen

⇨XÖV-DocBook-Mustervorlagen

⇨XÖV-XSD-Operationen

⇨XÖV-XSD-Vorlagen

⇨XÖV-UML-Profil

⇨XÖV-Werkzeug

XHamsterzucht

Fiktiver XÖV-Beispielstandard zur Veranschaulichung des möglichen Aufbaus eines UML-Modells.

⇨UML-Modell

⇨XÖV-Koordination

⇨XÖV-Standard

XMI

⇨XML Metadata Interchange

XML

⇨eXtensible Markup Language

XML Metadata Interchange

XML Metadata Interchange (XMI) ist ein Standard der OMG für den Austausch von UML-Modellen auf Basis von XML.

Link: http://www.omg.org/technology/documents/modeling_spec_catalog.htm#XMI

⇨eXtensible Markup Language

⇨Object Management Group

⇨UML-Modell

XML Schema Definition

XML-Schema Definition (XSD) ist eine Empfehlung des W3C zur Spezifikation syntaktischer Regeln für den Aufbau von XML-Dokumentstrukturen.

Link: http://www.w3.org/XML/Schema

⇨eXtensible Markup Language

⇨Syntax

⇨World Wide Web Consortium

XML-Schema

Datei, die den Regeln von XSD entspricht.

⇨XML Schema Definition

XÖV

XÖV steht für XML in der öffentlichen Verwaltung. Ziel von XÖV ist es, Datenaustausch innerhalb und mit der öffentlichen Verwaltung zu standardisieren. Hierdurch wird die Interoperabilität verbessert und in der Folge werden elektronische Prozesse einheitlicher und effizienter gestaltet.

⇨eXtensible Markup Language

XÖV-Basisdatentyp

Datentyp für die Werte-Einschränkung von Datenfeldern in XÖV-Standards, der von der XÖV-Koordination empfohlen wird. Die XÖV-Basisdatentypen umfassen die W3C-Datentypen für XSD sowie weitere XÖV-spezifische Datentypen, z.B. Code, String.Latin. Sie sind Teil des XÖV-UML-Profils.

⇨World Wide Web Consortium

⇨XML Schema Definition

⇨XÖV-Standard

⇨XÖV-UML-Profil

XÖV-Codeliste

Eine von der XÖV-Koordination empfohlene Codeliste mit fachunabhängiger Bedeutung, z.B. Geschlecht, die im XRepository veröffentlicht ist und in unterschiedlichen Standards eingesetzt werden kann.

⇨Codeliste

⇨XÖV-Koordination

⇨XRepository

XÖV-DocBook-Musteroperationen

Operation im Rahmen der XÖV-DocBook-Mustervorlagen.

⇨Operation

⇨XÖV-DocBook-Mustervorlagen

XÖV-DocBook-Mustervorlagen

Beispielhafte Templates zur Erzeugung von DocBook-XML-Dateien mit dem XGenerator.

⇨Template

⇨XGenerator

⇨DocBook

XÖV-Fachkomponente

XÖV-Fachkomponenten sind fachspezifische Datenbausteine, die in XOEV-Standards verwendet werden. Sie entstehen durch Einschränkung (Konkretisierung, Restriktion) einer XÖV-Kernkomponente. Aus einer XÖV-Kernkomponente koennen mehrere XÖV-Fachkomponenten abgeleitet werden. Die XÖV-Fachkomponenten sind im XRepository veroeffentlicht.

⇨XÖV-Kernkomponente

⇨XÖV-Standard

⇨XRepository

XÖV-Handbuch

Das XÖV-Handbuch enthält Anforderungen und Empfehlungen für die Entwicklung von XÖV-Standards.

⇨XÖV-Standard

XÖV-Invariante

Invariante im Kontext von XÖV zur Validierung der Einhaltung von XÖV-Namens- und Entwurfsregeln mit Bezug zum XÖV-UML-Profil.

⇨Invariante

⇨XÖV

⇨XÖV-Namens- und Entwurfsregeln

⇨XÖV-UML-Profil

XÖV-Kernkomponente

Im Rahmen von XÖV erstellte und durch die XÖV-Koordination herausgegebene Kernkomponente. Eine Kernkomponente ist ein fachuebergreifender Datenbaustein, der bei der Definition semantisch korrekter Informationseinheiten zum fachuebergreifenden Datenaustausch verwendet wird. Fuer die konkrete Anwendung der Kernkomponente in einem XÖV-Standard werden XÖV-Fachkomponenten gebildet, die die XÖV-Kernkomponente einschraenken. Die XÖV-Kernkomponenten sind im XRepository veroeffentlicht.

⇨XÖV-Fachkomponente

⇨XÖV-Standard

⇨XRepository

XÖV-Konformität

Die XÖV-Konformitaet stellt ein durch die XÖV-Koordination ausgestelltes Qualitaetsmerkmal eines XÖV-Standards dar, das die Einhaltung der XÖV-Konformitaetskriterien bescheinigt.

⇨XÖV-Konformitätskriterien

⇨XÖV-Koordination

⇨XÖV-Standard

XÖV-Konformitätskriterien

Die XÖV-Konformitaetskriterien sind die verbindliche Grundlage für die Pruefung der XÖV-Konformitaet eines Standards und zur dessen Einstufung als XÖV-Standard.

⇨Standard

⇨XÖV-Konformität

⇨XÖV-Standard

XÖV-Koordination

Die XÖV-Koordination ist eine Aufgabe der oeffentlichen Verwaltung, die unter anderem die XÖV-Konformitaet eingereichter Standards prueft. Sie bietet Information und Unterstuetzung für XÖV-Vorhaben waehrend des gesamten Lebenszyklus eines Standards.

⇨Standard

⇨XÖV-Konformität

⇨XÖV-Koordination

⇨XÖV-Vorhaben

XÖV-Metadaten

Beschreibende Merkmale zu Codelisten im XRepository und in Genericode-Dateien.

⇨Codeliste

⇨Genericode

⇨XRepository

XÖV-Modellierer

Person, die ein UML-Modell eines Standards bezueglich der XÖV-Konformitaetskriterien, insbesondere bezueglich der Anwendung des XÖV-UML-Profils, erstellt.

⇨Standard

⇨UML-Modell

⇨XÖV-Konformitätskriterien

⇨XÖV-UML-Profil

XÖV-Namens- und Entwurfsregeln

Die XÖV-Namens- und Entwurfsregeln legen die technische Ausgestaltung von XÖV-Standards fest. Die Einhaltung dieser Regeln ist ein wichtiges Kriterium zur Erlangung der XÖV-Konformitaet.

⇨XÖV-Konformität

⇨XÖV-Standard

XÖV-Produktionsprozess

Prozess mit den essentiellen Schritten zur Erstellung eines XÖV-Standards unter Anwendung des XÖV-Produktionszubehoers: Definition der Anforderungen, Erstellung und Export des UML-Modells, Validierung des exportierten UML-Modells, Generierung und Validierung der XML-Schemata sowie optional die Generierung und Validierung von DocBook-XML-Dateien.

⇨DocBook

⇨UML-Modell

⇨XML-Schema

⇨XÖV-Produktionszubehoer

⇨XÖV-Standard

XÖV-Produktionszubehoer

Das durch die XÖV-Koordination bereitgestellte XÖV-Produktionszubehoer stellt eine Sammlung von XÖV-Werkzeugen, Artefakten und Infrastrukturkomponenten dar, um einen XÖV-Standard entwickeln und betreiben zu können. Das XÖV-Produktionszubehör muss in eine vorhabenspezifische Produktionsumgebung integriert werden. Das XÖV-Produktionszubehoer allein ist zur Erstellung eines XÖV-Standards nicht hinreichend.

⇨Produktionsumgebung

⇨XÖV-Koordination

⇨XÖV-Standard

⇨XÖV-Werkzeug

XÖV-Projekt

Als XÖV-Projekt wird die Entwicklungsphase eines XÖV-Standards bezeichnet, es ist Teil eines XÖV-Vorhabens.

⇨XÖV-Standard

⇨XÖV-Vorhaben

XÖV-Standard

Als XÖV-Standard wird ein Standard bezeichnet, dessen XÖV-Konformitaet von der XÖV-Koordination festgestellt wurde.

⇨Standard

⇨XÖV-Konformität

⇨XÖV-Koordination

XÖV-UML-Modell

Ein UML-Modell, auf welches das XÖV-UML-Profil angewendet wurde.

⇨UML-Modell

⇨XÖV-UML-Profil

XÖV-UML-Profil

UML-Profil zur Erweiterung von UML im Kontext von XÖV, insbesondere zur spaeteren Erzeugung der XML-Schemata mit dem XGenerator.

⇨UML-Modell

⇨Unified Modeling Language

⇨XGenerator

⇨XML-Schema

⇨XÖV

XÖV-UML-Profil

Stereotyp des XÖV-UML-Profils.

⇨Stereotyp

⇨XÖV-UML-Profil

XÖV-Vorhaben

Im Rahmen eines XÖV-Vorhabens wird ein XÖV- Standard zunaechst entwickelt und dann betrieben. Das Vorhaben umfasst damit den gesamten Lebenszyklus des Standards.

⇨XÖV-Standard

XÖV-Werkzeug

Ein XÖV-Werkzeug ist ein von der XÖV-Koordination herausgegebenes Werkzeug, das den XÖV-Produktionsprozess zur Erstellung eines XÖV-Standards unterstuetzt.

⇨XÖV-Koordination

⇨XÖV-Produktionsprozess

⇨XÖV-Standard

XÖV-XSD-Operationen

Operation im Rahmen der XÖV-XSD-Vorlagen.

⇨Operation

⇨XÖV-XSD-Vorlagen

XÖV-XSD-Vorlagen

Templates zur XÖV-konformen Erzeugung von XML-Schemata mit dem XGenerator.

⇨Template

⇨XGenerator

⇨XML-Schema

⇨XÖV-Konformität

XPfleger

⇨XÖV-Modellierer

XRepository

Das XRepository ist eine von der XÖV-Koordination im Internet bereitgestellte, öffentlich zugängliche und zentral verwaltete Anwendung. Sie dient der Bereitstellung und Auffindbarkeit von Datenaustauschformaten, insbesondere für XÖV-Inhalte wie XÖV-Kernkomponenten, XÖV-Standards, XÖV-Codelisten, XÖV-Fachkomponenten und Codelisten dort bereitgestellt.

Link: www.xrepository.de

⇨Codeliste

⇨XÖV-Codeliste

⇨XÖV-Fachkomponente

⇨XÖV-Kernkomponente

⇨XÖV-Standard

XSD

⇨XML Schema Definition

D Versionshistorie

In diesem Anhang werden je Release des XÖV-Handbuchs die Bereiche beschrieben, in denen Veränderungen oder Ergänzungen vorgenommen worden sind.

D.1 Release 1.1

CR 2011-204: Anpassung des XÖV-Konformitätskriteriums K-4 (Veröffentlichung)

Die Regel wurde bzgl. des Ablaufs angepasst. Die Veröffentlichung eines Standards im XRepository ist keine Vorbedingung mehr, statt dessen muss die Veröffentlichung unverzüglich nach der Zertifizierung erfolgen.

CR 2011-09-06-1: Änderung am Codelistenformat (Genericode)

Die Kennzeichnung einer Spalte der Codelisten-Repräsentation als 'codename' (gemäß XÖV-Basisdatentyp Code, Attribut "name") wurde entfernt. Grund: In der Praxis der Darstellung von Codelisten hat sich diese Einschränkung als hinderlich erwiesen. Codelisten sind rationeller darstellbar und vielseitiger einsetzbar, wenn man solche Festlegungen dem User (Einsatz der Codeliste im XÖV-Standard oder in einer Applikation) überlässt.

CR 2010-011: Aufnahme weiterer Zeichen in String.Latin und Veröffentlichung eines Standards "Lateinische Zeichen in Unicode"

Der Datentyp String.Latin wird zukünftig durch einen eigenständigen Standard *“Lateinische Zeichen in Unicode”* (verfügbar unter http://xoev.de/latinchars/) beschrieben. Dieser Standard beschreibt den Zeichensatz der lateinischen Zeichen sowie den entsprechenden Datentyp String.Latin in XML-Schema und UML. Gegenüber der in der Version 1.0 des XÖV-Handbuchs veröffentlichten Fassung von String.Latin wurden Zeichen ergänzt. Für die inhaltlichen Änderungen wird auf den genannten Standard verwiesen.

CR 2010-013 / CR 2011-06-16-6: Unterstützung zusätzlicher Eigenschaften bei Wildcard Elementen

Der Stereotyp xsdAnyContents unterstützt nun die Angabe der Häufigkeiten (minOccurs, maxOccurs) und die Art der Inhaltsverarbeitung (processContents).

CR 2011-06-16-4: Modellierung von Unions

Mit Hilfe des Stereotypen xsdUnion können jetzt xs:union Datentypen modelliert werden.

CR 2011-06-17-1: Restriction base bei Enumerations

Für Enumerationen im Schema kann ein beliebiger einfacher Typ als Basistyp angegeben werden.

CR 2011-06-16-3: Globale Elemente mit Choice

Globale Elemente können als Inhaltsmodell nun auch einen Choice haben.

CR 2011-06-16-2: Referenzierung von globalen Elementen, Element- und Attributgruppen

Die Referenzierung von globalen Elementen, Element- und Attributgruppen ist jetzt möglich.

CR 2011-06-16-1: Typisierte globale Elemente

Globale Elemente können anhand eines komplexen Typen definiert werden.

CR 2011-06-17-2: Element- und Attributgruppen

Element- und Attributgruppen können definiert werden.

CR 2011-05-20-1: Änderung der Angaben zum XMI-Formats des XGenerators

Der XGenerator liest ab der Version 2.2.0 nur noch XMI-Dateien in der Version 2.x und keine 1.X Dateien mehr.

CR 2010-102, 2010-103, 2010-10-01-1: Dokumentation von neuen Elementen für die Dokumentationsgenerierung

Das XOEV-Profil wurde um verschiedene Elemente erweitert, die bei der Dokumentationsgenerierung berücksichtigt werden können. Dabei handelt es sich um neue Stereotypen (xsdDeprecated, xsdXModelImport) und neue Eigenschaften von Stereotypen (xsdCode.documentationCodeList)

CR 2011-09-06-1: Entfernen der Invarianten

Die technische Umsetzung der Namens- und Entwurfsregeln durch Invarianten wird im Handbuch nicht mehr im Detail beschrieben.

Index

A

Anzeigepflicht 17

B

Basisdatentyp 9, 19
Code 56
W3C 25
Beispielprojekt
im XRepository 10
Bundesanzeiger 79

C

Code 56
Abweichung von CCTS 58
unterschiedliche Arten der Nutzung 92
Code (Datentyp) 9
Codeliste 19
Genericode Format 84
Haftung 89
Historische Einträge 85
Integration in den Standard 91
schemavalidierend 90
Staatenschlüssel 92
Unscharf 98
Werkzeugunterstützung 88
XÖV-Metadaten 87

D

Docbook 7, 24, 135, 148
DTD 151
Dokumentation
HTML 151
PDF 151
DSMeld 80
DVDV 20

E

EMF 138

F

Fachkomponente 19

G

Genericode 84
Genericoder 88

H

Haftung 89
Herstellerunabhängigkeit 15

I

Invariante 27, 134

J

Java 136

K

Kernkomponente 19
Bibliothek im XRepository 9
Konformitätskriterien
Verbindlichkeit 13
Konformitätsprüfung 4
XGenerator 19

L

Lebenszyklus 4

M

Modell
konzeptionell und physisch 25

N

Namens- und Entwurfsregeln 8
Namensraum 76

O

Object Constraint Language 27
OCL 27, 134, 158
Öffentliche Verwaltung
Entscheidung über Entwicklungen 15
OSCI-Transport 20

P

PKI-1 Verwaltung 20
Produktionszubehör 4

R

Rechte (am Standard) 15
Rechtsnachfolge von Behörden 85
Redundante Entwicklungen 17
Religion
als Codeliste 80

S

Staaten- und Gebietsschlüssel (Codeliste) 92
Statistisches Bundesamt 80
Steckbrief 4
SVG 148

U

UML 17
Aktivitätsdiagramme 17
Invariante 27
Metamodell für XÖV-Standards 8
Stereotyp 27
Tagged Value 27
XMI 18
UML Modell
Baukasten 65
Struktur 65
UML Profil 27

V

Velocity 135, 157

W

Wiederverwendung 8, 19
Codeliste 79
WSDL Datei 26

X

XGenerator 7, 17
Encoding 139
Funktionsweise 132
XMI 18, 138
XML Schema
Grafik 148
Namespace 76
XÖV
Konformitätsprüfung 4
Produktionszubehör 4
Profil 8
Steckbrief 4
XPfleger 24
XRepository 7
Pflicht zur Veröffentlichung 16

Z

Zeitzone 54