Inhaltsverzeichnis

1 Einführung

Die Kommunikation zwischen Geschäftspartnern und der Austausch von Informationen zwischen eben diesen stellt eine große Herausforderung für alle Beteiligten dar, die häufig mit hohen Kosten und einer mitunter enormen Komplexität verbunden sind.

Gerade systemübergreifende Geschäftsprozesse – auch über Unternehmensgrenzen hinweg – erfordern, dass alle beteiligten Systeme die prozessrelevanten Daten miteinander in geeigneter Form austauschen können und insbesondere bezüglich der Daten ein gleiches Verständnis haben.

In der Folgezeit stieg damit auch die Bedeutung der Extensible Markup Language (XML) im Umfeld der Sozialversicherung.

Die Vorteile von XML sind nicht zuletzt darin zu sehen, dass XML als offener Internetstandard eine standardisierte, textbasierte Meta-Auszeichnungssprache darstellt, die es ermöglicht, Daten bzw. Dokumente bezüglich Inhalt und Darstellungsform derart zu beschreiben und zu strukturieren, dass sie – vor allem auch über das Internet – zwischen einer Vielzahl von Anwendungen in verschiedensten Hard- und Softwareumgebungen hersteller- und branchenneutral automatisiert ausgetauscht und weiterverarbeitet werden können.

Durch die Schaffung einer einheitlichen XML-Empfehlung sollen sämtliche XML-Aktivitäten im Umfeld der Sozialversicherung zentralisiert und gebündelt werden. Durch deren modularisierten Aufbau sollen so zukünftig einheitliche und integrierte XML-Schnittstellen entwickelt werden können. Dies kann mittel- und langfristig zu Effizienzsteigerungen und Kosteneinsparungen führen. Insbesondere lassen sich durch dieses Vorgehen die Implementierungsaufwände drastisch reduzieren.

Durch die XML-Empfehlung wird für den Datenaustausch auf Basis von XML ein standardisiertes und einheitliches Rahmenwerk geschaffen, mit dessen Hilfe alle zukünftigen XML-Schnittstellenimplementierungen vollständig beschrieben werden können. Hierbei werden nicht nur die Sprachelemente und konkreten Entwurfsprinzipien vorgeschrieben, sondern auch die Grundstrukturen verfahrensneutral festgelegt.

2 Allgemeine Grundlagen

2.1 Verwendung von Schlüsselworten

Für die genaue Unterscheidung zwischen der Verbindlichkeit und Aussagekraft von Inhalten und Vorgaben werden die dem [RFC 2119] entsprechenden Großbuchstaben in deutscher Sprache verwendet:

- **MUSS** bedeutet, dass es sich um eine absolut gültige Festlegung bzw. Anforderung handelt.
- **DARF NICHT** bezeichnet den absolut gültigen Ausschluss einer Festlegung bzw. Anforderung.
- **SOLL** beschreibt eine Empfehlung. Abweichungen zu diesen Festlegungen sind in begründeten Fällen möglich.
- **SOLL NICHT** kennzeichnet die Empfehlung, eine Eigenschaft auszuschließen. Abweichungen sind in begründeten Fällen möglich.
- **KANN** bedeutet, dass die Eigenschaften fakultativ oder optional sind und damit keinen allgemeingültigen Empfehlungscharakter besitzen.

2.2 Begriffsdefinition

Dieses Kapitel dient der Definition und Festlegung allgemeiner Begriffe.

Definition:

XML-Regelwerk – Allgemeine Entwurfsprinzipien und Vorgaben zur Implementierung einer auf der XML-Empfehlung basierenden XML-Schnittstellenimplementierung.
(vgl. Kapitel 5 – Entwurfsprinzipien)

XML-Empfehlung – Die XML-Empfehlung enthält neben dem XML-Regelwerk auch noch Vorschriften zur Versionierung von Schema-Dateien sowie Regeln zur Vergabe von Schemaversionsnummern.

XML-Schnittstelle – Ein offenes und standardisiertes Datenformat, das einen automatisierten elektronischen und bidirektionalen XML-Datenaustausch zwischen allen Kommunikationspartnern im SV- Umfeld ermöglicht, erfordert die Definition von Semantik, Struktur und von Datentypen in Form von XML-Schemata für die Implementierung einer auf der XML-Empfehlung basierenden XML-Schnittstelle.

Im Gegensatz zu der XML-Empfehlung, die sich auf einem höheren Abstraktionsniveau befindet, stellt eine XML-Schnittstelle eine konkrete Anwendung, d.h. Umsetzung und Ausprägung, der XML-Empfehlung dar.

2.3 Positionierung und Geltungsbereich

Die XML-Empfehlung stellt eine konkrete technische Implementierung und Anwendung von XML-Standards für die Nutzdaten in den Datenaustauschverfahren dar.

Das Dokument beschreibt die allgemeinen Anforderungen und die normativen Festlegungen für die technische Umsetzung der XML-Empfehlung. Insbesondere werden verbindliche und einheitliche Regeln sowie Festlegungen für den Schemaentwurf getroffen. Die Darstellung und Beschreibung von Detailinformationen zu XML-Schema ist nicht vorrangiges Ziel dieses Dokuments. Hier wird an gegebener Stelle auf entsprechende Sekundärliteratur und Lehrbücher verwiesen.

3 Anforderungen und Zielsetzung

3.1 Beschreibung

Der Nutzung von XML für den strukturierten Austausch von Daten im SV-Umfeld kommt eine immer größere Bedeutung zu. Insbesondere existiert eine Vielzahl von neuen Verfahren im Leistungserbringerbereich, die auf Basis von der XML-Empfehlung implementiert werden könnten.

Aus konkreten Implementierungen können die Vorgaben und Festlegungen der XML-Empfehlung anhand von Praxiserfahrungen im Zeitablauf evaluiert und somit gehärtet bzw. falsifiziert oder aber auch ergänzt werden. Über einen solchen dynamischen Prozess kann die Qualität der XML-Empfehlung stetig überprüft und im Zeitablauf verbessert werden.

Originäres Ziel MUSS es sein, sämtliche XML-Aktivitäten im Umfeld der Sozialversicherung zu vereinheitlichen und in Form eines Rahmenwerkes verbindlich für die Implementierung von Schnittstellen festzuschreiben.

Der Datenaustausch auf Basis des etablierten DTA- Verfahrens erfolgt auf eine integrierte, einheitliche und konsistente Art und Weise.

Auch der Datenaustausch von neuen Verfahren auf Basis von XML MUSS zukunftssicher abgebildet werden. Neben Wirtschaftlichkeitsaspekten und Fragen der organisatorischen Machbarkeit erfordert dies nicht zuletzt ein organisatorisches und technisches Rahmenwerk, dessen Grundlage, durch die Ausgestaltung der einheitlichen XML-Empfehlung für die Sozialversicherung, der sog. GI4X, sein MUSS. Bestehende DEÜV- Verfahren bleiben hiervon unberührt.

In diesem Zusammenhang MÜSSEN auch die aktuellen Entwicklungen zum Thema XML, insbesondere im direkten Umfeld der Sozialversicherung, betrachtet und gegebenenfalls berücksichtigt werden.

Eine klare Positionierung und Abgrenzung von der XML-Empfehlung stellt eine unabdingbare Voraussetzung für die Etablierung und Anerkennung dar.

3.2 Formalisierung

Basierend auf der zuvor beschriebenen Zielsetzung ergeben sich folgende zielgerichtete und verbindliche Handlungen:

- Bewertung und Analyse relevanter Standards und Spezifikationen.
- Festlegung von Regeln zur Vergabe von Dateinamen.
- Verwendung von XML-Schema für die Modellierung von der XML-Empfehlung.

- Beschreibung aller zulässigen Schemasymbole und Beschränkung der Sprachelemente von XML-Schema.

- Festschreibung der Namensraumtopologie, Präfixe und Abhängigkeiten.

- Festlegung allgemeiner Entwurfsprinzipien für die Erstellung verfahrensspezifischer Nutzdatenteile.

- Festlegung von Regeln zur Schemaversionierung.
- Festlegung von Vorgaben für die Ableitung neuer XML-Schnittstellen zur Vermeidung von Komplexität und Wildwuchs bei der Erstellung von XML-Schnittstellen im Umfeld der SV.

3.3 Ausblick

Das Ziel dieses Dokumentes ist es, eine solide Basis für weiterführende Standardisierungen im XML-Umfeld zu schaffen – nicht jedoch diese bereits vorwegzunehmen.

Daher wurde in diesem Dokument bewusst darauf verzichtet, verfahrensübergreifende funktionale Datenstrukturen oder organisatorische Strukturen festzulegen oder zu standardisieren. Es wird jedoch darauf hingewiesen, dass eine Vereinheitlichung derselben eine wesentliche Grundlage für den effektiven, homogenen, und erfolgreichen Einsatz von XML-Schnittstellen bildet.

Die dringendsten zukünftigen XML-Standardisierungsbemühungen sollten nach Meinung der Arbeitsgruppe, die dieses Dokument erarbeitet hat, in folgende Richtungen gehen:

- Bereitstellung eines GI4X Basis-schemas mit verfahrensübergreifenden Datentypdefinitionen.

- Bereitstellung eines verfahrensübergreifenden Transportformates, das neben den verfahrensspezifischen Nutzdaten eine einheitliche Modellierung von Transportmetadaten vorsieht – ähnlich zu dem heute existierenden Auftragssätzen.

- Bereitstellung eines verfahrenübergreifenden Formates für Verarbeitungs- und/oder Fehlerprotokolle, sowie Fehlerkataloge.

- Bereitstellung eines verfahrensübergreifenden Formates für die Modellierung von Schlüsseltabellen.

- Bereitstellung eines Repository-Formates zur Modellierung der notwendigen Meta-Informationen für Verfahren, wie verwendete Schemata, Schlüsseltabellen, und Fehlerkataloge, sowie ein standardisiertes Archiv für diese Meta-Daten, Schema-Dateien und Schlüsseldateien, das sowohl lokal oder aber über Internet zugreifbar ist.

4 Dateinamenskonvention

Dieses Kapitel dient dazu, eine verbindliche und verfahrensübergreifende Dateinamenskonvention für XML-Schema-Dokumente und XML-Schlüsseltabellen zu definieren, da diese eine wesentliche Grundlage für den Austausch und die Validierung verfahrensspezifischer Nutzdaten darstellen.

Die Benennung von XML-Nutzdatendateien[1] ist von dieser Konvention unberührt und wird weiterhin verfahrensspezifisch geregelt werden.

4.1 Dateinamen

Dateinamen dürfen aus Groß- und Kleinbuchstaben sowie Unterstrich ´_´, Punkt ´.´, und Bindestrich ´-´(Minus) bestehen. Der Bindestrich ist als Trennelement reserviert. Umlaute und ‚ß' DÜRFEN in Dateinamen NICHT verwendet werden, da sie nicht URL konform sind. Für Namen SOLLEN vorrangig deutsche Begriffe verwendet werden. Bereits aus dem Englischen in den Duden übernommene Begriffe wie ‚Plug-in' können aber beibehalten werden. Dateinamen besitzen eine vorgegebene Grundstruktur, die bei der Vergabe von Dateinamen zu berücksichtigen ist. Die folgenden Komponenten sind Teil der Grundstruktur:

[VK] Verfahrenskennung.

Die Verfahrenskennung ist verpflichtend und beschreibt die logische Zugehörigkeit der Datei zu einem bestimmten Verfahren oder Verfahrenslandschaft. Die Verfahrenskennung „GI4X" ist für zentrale, d.h. alle Verfahren betreffende, Dateien reserviert. Darüber hinaus können durch Koordinationsgremien weitere verfahrensübergreifende Kennungen verabschiedet werden.

Verfahrensspezifische Kennungen besitzen den Aufbau/Name wie in der Anlage 4 der Gemeinsamen Grundsätze Technik definiert und werden verfahrensintern festgelegt.

Beispiel: EMDP0 (Verfahrensspezifisch: Echtdaten-Datenaustausch „MDK Bereich Pflege")

[QN] Qualifizierender Name.

Dieser Bestandteil ist verpflichtend und dient zur weiteren Unterscheidung von Dateien im Kontext einer Verfahrenskennung. Der QN darf dabei aus Gross- und Kleinbuchstaben sowie Unterstrich ´_´ bestehen.

[VN] Versionsnummer.

[1] Zu diesen zählen die verfahrensspezifischen Datenlieferungen, nicht jedoch von diesen Dokumenten referenzierte Schlüsseltabellen!

Dieser Bestandteil ist verpflichtend und enthält die Versionsnummer der Datei, wobei die in dieser Empfehlung vorgegebenen Versionierungsvorschriften das Format der Versionsnummer vorgeben.

[LN] Laufende Nummer.

Dieser Bestandteil ist optional und dient dazu, Dateien zu identifizieren, die von einem ‚Master' Dokument (das keine [LN] Komponente hat) importiert/inkludiert/referenziert werden. Laufende Nummern beginnen bei 1.

[SUF] Suffix.

Dieser Bestandteil ist verpflichtend und beschreibt den Typ der Datei: „xml" für Schlüsseltabellen und „xsd" für XML-Schema-Dokumente.

Der Dateiname ergibt sich aus diesen Komponenten wie folgt: [VK]-[QN]-[VN]-[LN].[SUF]

D. h. bis auf [SUF] werden die Komponenten durch Bindestrich getrennt, der Suffix durch Punkt von den anderen Komponenten.

Hinweis: Die [VK] Komponente soll die Unterscheidung zwischen (E) Echt- und (T) Testdaten betrieb unterstützen.

4.2 Schema-Dateien (XSD)

Insbesondere für XSD-Dateien, MUSS eine verbindliche Dateinamenskonvention vorgegeben und eingehalten werden. [VK], [QN], [VN], und [SUF] sind verbindliche (MUSS) Namensbestandteile, [LN] ist optional (KANN). Die Versionsnummernkomponente [VN] besteht aus der Haupt-, Neben-, und Revisionsnummer der Schemaversion. Der [QN] ‚basis' ist für verfahrensübergreifende oder verfahrensspezifische Schema-Definitionen reserviert, die von mehreren Schemata importiert werden:

Dateiname	Art	Beschreibung
GI4X-basis-[VN].xsd	MUSS	GKV Basis Schema: Dieses Schema enthält den kleinsten gemeinsamen Nenner an GKV weit zu verwendenden Schemadefinitionen (einfache/komplexe Typen, Elemente und Attribute), d. h. Definitionen die für alle Verfahren relevant sind.
[VK]-basis-[VN].xsd [VK]-[QN]-[VN].xsd	KANN	Verfahrensübergreifendes/-spezifisches Basis Schema: Die so benannten Dateien enthalten gemeinsam genutzte Definitionen, die von anderen Schemata importiert werden – aber nur im Kontext eines Verfahrens oder einer Verfahrenslandschaft, z. B. MDK, Leistungserbringer oder Arbeitgeber sinnvoll sind.

		Neben dem QN ‚basis' können für verfahrensübergreifende Schemata auch noch andere QN Bezeichner verwendet werden.
[VK]-[QN]-[VN].xsd	MUSS	Verfahrensspezifisches Schema: Diese Schemata enthalten Nachrichtentyp-spezifische Schemadeklarationen für ein Verfahren (einfache und komplexe Typen, Elemente und Attribute). Die QN Komponente sollte dabei den Nachrichtentyp beschreiben.

Tabelle 1: Dateinamenskonvention für XSD-Dateien

Beispiele:

1. Die Datei ‚EAPO0-ABRP-1.2.0.xsd' enthält das fachspezifische Verfahrensschema Echtzeitbetrieb für APO, Nachrichtentyp ABRP, in der Schemaversion 1.2.0. Die Datei ‚EAPO0-ABRP-1.2.0-1.xsd' ist eine Subschemadatei dieses Schemas mit laufender Nummer 1, das von dem Hauptschema inkludiert wird.

2. Zwischen Krankenkassen und MDK (Medizinischer Dienst der Krankenkassen) werden demnächst zwei XML-Verfahren eingeführt: MDK-Krankenhaus und MDK-Pflege. Diese haben die Verfahrenskennungen EMDK0/TMDK0 und EMDP0/TMDP0. In beiden Verfahren werden zum Teil dieselben Schemadefinitionen benötigt. Um diesem Umstand gerecht werden zu können und redundante Schemateile zu vermeiden, könnte ein MDK Koordinationsgremium als verfahrensübergreifende [VK] für Dateinamen ‚MDK´ einführen. Die von beiden Verfahren gemeinsam zu nutzenden Schemadefinitionen könnten dann in einer ‚MDK-basis-1.0.0.xsd' Schemadatei verfahrensübergreifend definiert werden.

D. h. für Beispiel 2 könnte sich die folgende Import Hierarchie ergeben:

- ‚GI4X-basis-1.0.0.xsd' wird von allen MDK Schemadateien importiert.
- ‚MDK-basis-1.0.0.xsd' wird von allen MDK verfahrensspezifischen Schemadateien importiert, z. B. den Schemadateien ‚EMDK0-Beauftragung-1.0.0.xsd' und ‚EMDP0-Beauftragung-1.0.0.xsd'.

Eine Ausnahme von dieser Dateinamenskonvention gilt für sogenannte ‚Brücken' Schemata. Brücken Schemata werden notwendig, wenn man komplexe Typ-Definitionen aus einem Schema für ein anderes Schema (mit einem anderen Namensraum) einschränken will. Dies ist im XML-Schema nicht direkt im Zielschema möglich, sondern muss in einem Schema erfolgen, das denselben Zielnamensraum wie die komplexen Typen besitzt, die eingeschränkt werden sollen. Konzeptuell gehören diese eingeschränkten Typen somit zum Zielschema, vom Namensraum her aber zum Schema, das die Basistypen definiert.

Bei der Benennung solcher Brücken Schemata wird dem Umstand, dass die enthaltenen Definitionen zu 2 Schemata gehören, durch ein Benennungsschema Rechnung getragen, das beide Schemata berücksichtigt:

[VK1]-[QN1]-[VN1]--[VK2]-[QN2]-[VN2].xsd

Der erste Teil „[VK1]-[QN1]-[VN1]“ des Dateinamens MUSS dabei der Dateiname (ohne Suffix) des Schemas sein, aus dem die inkludierten Basistypen des Brückenschemas stammen. Der zweite Teil des Dateinamens [VK2]-[QN2]-[VN2] MUSS dabei der Dateiname (ohne Suffix) des Schemas sein, das die eingeschränkten Typen verwendet (importiert). Der Trenner zwischen den beiden Bestandteilen sind zwei ‚-‘, Bindestriche.

4.3 Schlüsseltabellen

Obwohl das Austauschformat von Schlüsseltabellen in der Empfehlung nicht weiter spezifiziert wird, werden externe Schlüsseltabellen bei der Modellierung von Identitätsconstraints (s. Kapitel 5.5.9) explizit eingeführt und daher ist es nötig, Konventionen für die Benennung von Schlüsseltabellendateien zu definieren.

Die Benennung von Schlüsseltabellen erfolgt nach den allgemeinen Dateinamenskonventionen. Die Komponenten [VK], [QN], [VN], und [SUF] sind verpflichtend, [LN] darf nicht verwendet werden. Der Qualifizierende Name [QN] enthält eine Beschreibung des Inhaltes der Schlüsseltabelle und muss mit ´_keys’ enden. Die Versionsnummer [VN] einer Schlüsseltabelle ist nicht strukturiert und besteht aus einer einfachen fortlaufenden Nummer [1...n].

Das folgende Beispiel nimmt an, dass GI4X verfahrensübergreifend Schlüsseltabellen für Betriebsnummern und Institutskennzeichen modelliert. Die zugehörigen Schlüsseltabellennamen könnten wie folgt vergeben werden:

GI4X-BN_keys-1.xml	1. Version der Liste von gültigen Betriebsnummern
GI4X-IK_keys-2.xml	2. Version der Liste von gültigen Institutskennzeichen

5 Entwurfsprinzipien

5.1 Überblick

Aus Gründen der Komplexitätsvermeidung und Vereinheitlichung MUSS die Menge an zulässigen Modellierungsmöglichkeiten vom XML-Schema eingeschränkt werden.[2] Die nachfolgend beschriebenen Vorgaben sind verbindlich und bei der Erstellung von konkreten XML-Schnittstellenimplementierungen zu berücksichtigen.

Diese Vorgaben bilden den konzeptionellen und technischen Rahmen für die Ableitung und Erstellung zulässiger und konformer Schnittstellen auf Basis der XML-Empfehlung.

Letztendlich wird damit auch das Ziel verfolgt, den Umfang und die relative Komplexität vom XML-Schema auf eine effektive Teilmenge der vom XML-Schema bereitgestellten Funktionen zu begrenzen und somit einheitliche Designvorgaben für den Schnittstellenentwurf im Umfeld der Sozialversicherung verbindlich festzuschreiben.

5.2 Namensraumhierarchie

Es MÜSSEN Namensräume und Präfixe explizit verwendet werden. Namensräume dienen der Bildung abgegrenzter Gültigkeitsbereiche für Elemente und Attribute, um so die Nutzung beliebiger XML-Vokabulare in einem Dokument zu ermöglichen.

Damit kann die Wiederverwendung bestehender XML-Vokabulare berücksichtigt und so die Modellierung neuer Schnittstellenimplementierungen vereinfacht und damit erleichtert werden.

Die daraus resultierende Namensraumhierarchie wird anhand der nachfolgenden Abbildung näher dargestellt:

[2] vgl. Kapitel 5.5.2- Allgemeine Festlegungen

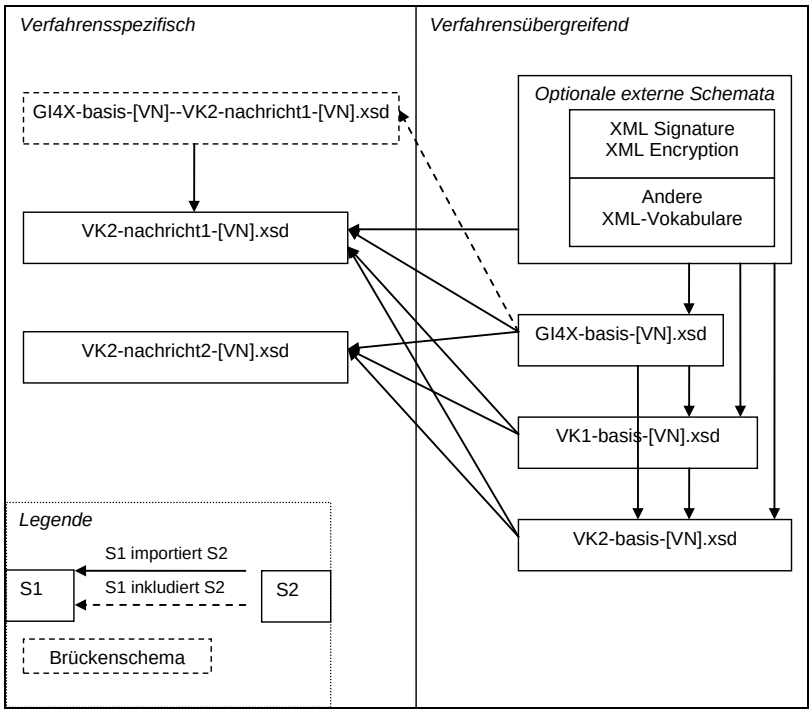


Abbildung 1: Darstellung der Namensraumhierarchie

Jedes Schema in der obigen Abbildung definiert dabei einen eindeutigen Schema-Namensraum wie im Folgenden beschrieben. Durch den Import werden die Definitionen der importierten Schemata in den importierenden Schemata verfügbar und können dort über qualifizierte Namen (lokale Namen und Präfixe, die an Namensräume gebunden sind) referenziert werden.

Hinweis: Im obigen Beispiel importiert ‚VK2-nachricht1-[VN].xsd' externe Schemata, nicht jedoch ‚VK2-nachricht2-[VN].xsd'. Der Import von Schemata ist bedarfsgetrieben und daher müssen nicht benötigte Definitionen auch nicht importiert werden.

Festlegung: Jeder Namensraum MUSS eindeutig innerhalb der Namensraumhierarchie von GI4X sein.

Festlegung: Im jeweiligen Instanzdokument MÜSSEN die verwendeten Namensräume im Root-Element angegeben werden. Eine andere Lokation ist unzulässig.

Die Namensraumbenennung MUSS folgenden allgemeinen Aufbau besitzen, der insbesondere für konkrete Schnittstellenimplementierungen verbindlich vorgeschrieben und damit zu berücksichtigen ist: GI4X:/xml-schema/[VK]-[QN]/[VN']

[VK],[QN]: Bedeutung wie in Kapitel 4 definiert.
[VN] : Versionsnummer (mit Revisionsnummer-Komponente) des Schemas.
[VN'] : Versionsnummer (ohne Revisionsnummer-Komponente) des Schemas.

Eine Ausnahme bilden sogenannte Brücken Schemata (s. 4.2). Der Namensraum eines Brückenschemas MUSS derselbe sein wie der des Schemas, das die in einem Brückenschema eingeschränkten Typen definiert.

Namensraum	Präfix	Art	Schema
GI4X:/xml-schema/GI4X-basis/[VN']	GI4X-basis	MUSS	GI4X-basis-[VN].xsd GI4X-basis-[VN1]--[VK2]-[QN2]-[VN2].xsd
GI4X:/xml-schema/[VK]-basis/[VN']	[VK]-basis	MUSS	[VK]-basis-[VN].xsd [VK1]-basis-[VN1]--[VK2]-[QN2]-[VN2].xsd
GI4X:/xml-schema/[VK]-[QN]/[VN']	[VK]-[QN]	MUSS	[VK]-[QN]-[VN].xsd [VK1]-[QN1]-[VN1]--[VK2]-[QN2]-[VN2].xsd

Tabelle 2: Darstellung der Namensraumhierarchie

5.3 Bezeichner für Schema- Elemente

Die einheitlichen Entwurfsprinzipien beziehen auch die Bezeichner für Schema-Elemente ein. Als Bezeichner sind die Namen der Schema- Elemente zu verstehen.

Festlegung: Alle Bezeichner SOLLTEN **deutsche Wörter** (auch Zusammensetzungen aus mehreren Wörtern) verwenden. Abkürzungen SOLLEN weitestgehend vermieden werden. Als Ausnahme KÖNNEN (verfahrens-) bekannte Abkürzungen, wie beispielsweise BBNR, PLZ verwendet werden.

Die Namen (Identifikator) für Gruppen, Typen, Listen, Enumerationen, Union und Identitätseinschränkungen enden mit dem jeweiligen Schema- Elementnamen, der jedoch auf maximal 3 Zeichen beschränkt wird.

Festlegung: Die Bezeichner SOLLTEN kurze, aussagekräftige Namen haben.

Festlegung: Der zulässige Zeichenvorrat für Bezeichner MUSS eingeschränkt werden. Zulässig sind nur folgende Zeichen:

[0-9]: Numerische Zeichen
[A-Z,a-z]: Buchstaben des deutschen Alphabets ohne Umlaute und ß
[_]: Als Trennzeichen ist nur _ zulässig.

Festlegung: Schema- Elemente DÜRFEN NICHT mit numerischen Zeichen beginnen.

Festlegung: Ein Identifikator zur Unterscheidung der einzelnen Schema-Elemente MUSS verwendet werden.

In der nachfolgenden Tabelle werden diese rudimentären Vorgaben näher beschrieben:

Schema-Element	Festlegung	Art	Beispiel
Element	Elementnamen beginnen immer mit einem Großbuchstaben.	MUSS	<xs:element name="Name">
Attribut	Attributnamen dürfen keine Großbuchstaben verwenden.	MUSS	<xs:attribute name="name">
Attributgruppe	Attributgruppennamen unterliegen den Restriktionen für Attributnamen und enden mit dem Identifikator _Grp.	MUSS	<xs:attributeGroup name="name_Grp">
Modellgruppe	Modellgruppennamen beginnen immer mit einem Großbuchstaben und enden mit dem Identifikator _Grp.	MUSS	<xs:group name="Name_Grp">
Einfache und komplexe Typen	Die Namen von einfachen - und komplexen Typen unterliegen den Restriktionen für Elemente. Einfache Typen enden zusätzlich mit dem Identifikator _Stp, komplexe Typen mit _Ctp. Eine Ausnahme bilden Listen und Vereinigung, die statt mit _Stp mit den Identifikatoren _Lst bzw. _Unn enden.	MUSS	<xs:simpleType name="Name_Stp"> <xs:complexType name="Name_Ctp">
Listen	Der Name von Listen unterliegt den Restriktionen für Elemente und endet zusätzlich mit dem Identifikator _Lst.	MUSS	<xs:simpleType name="Name_Lst"> <xs:list itemType="xs:string"/> </xs:simpleType>
Vereinigung	Der Name von Vereinigungen unterliegt den Restriktionen für Elemente und endet zusätzlich mit dem Identifikator _Unn	MUSS	<xs:element name="Dateigroesse_Unn"> <xs:simpleType> <xs:union memberTypes="Union1_Stp Union2_Stp" /> </xs:simpleType> </xs:element>
Identitäts-einschränkung	Für Identitätseinschränkungen gelten die Restriktionen für Elemente in Verbindung mit folgenden Restriktionen: **<xs:unique>** - Der Name endet mit dem Identifikator _Uqe. **<xs:key>** - Der Name endet mit dem Identifikator _Key. **<xs:keyref>** - Der Name endet mit dem Identifikator _Krf.	MUSS	<xs:unique name="Name_Uqe"> <xs:selector xpath="Ausgangsmenge" /> <xs:field xpath="eindeutiger Wert" /> </xs:unique> <xs:key name="Name_Key"> <xs:selector xpath="Ausgangsmenge"/> <xs:field xpath="eindeutiger Wert " /> </xs:key> <xs:keyref name="Name_Krf" refer=" Name_Key"> <xs:selector xpath="Ausgangsmenge" /> <xs:field xpath=" eindeutiger Wert " /> </xs:keyref>

Tabelle 3: Bezeichner für Schema-Elemente

5.4 Zeichensatz

Für den Datenaustausch im Gesundheits- und Sozialwesen sind die zulässigen Zeichensätze in der Anlage 15 der Gemeinsamen Grundsätze Technik geregelt und entsprechend zu berücksichtigen.

5.5 XML-Schema - Modellierungsvorgaben

5.5.1 Überblick

Der W3C-Standard [XML-Schema] ist eine XML-Sprache zur Definition von XML-Vokabularen.
Als Nachfolger der bekannten Document Type Definitions (DTD) markiert XML-Schema den entscheidenden Wendepunkt von der bisherigen, dokumentenorientierten hin zu einer datenorientierten Sichtweise.

Technisch gesehen erweitern sich durch XML-Schema die Ausdrucksmöglichkeiten bei der Formulierung neuer XML-Vokabulare entscheidend. Einerseits hinsichtlich inhaltlicher Merkmale wie die Typisierung von Daten, andererseits auch in Richtung struktureller Merkmale. So halten bekannte Konzepte aus den Programmier- und Datenbanksprachen Einzug. Damit ergeben sich weitaus flexiblere Datenmodellierungsmöglichkeiten. Insbesondere werden objektorientierte Mechanismen, wie Vererbung und Wiederverwendung, unterstützt. Somit können sowohl die Struktur als auch die Daten selbst beschrieben werden.

XML-Schema stellt eine Beschreibungssprache für XML-Dokumente dar und bildet somit die technische Grundlage für die Definition und Implementierung von XML-Schnittstellen. Dabei stellen XML-Schemata selbst XML-Dokumente dar.

Hinweis: XML-Schema unterscheidet zwischen der **Deklaration** (Komponenten, die in dem Instanzdokument sichtbar sind, wie beispielsweise Elemente, Attribute und Notationen) und der **Definition** (interne Komponenten, die in dem Instanzdokument nicht sichtbar sind, wie etwa Typen).

In dem nächsten Kapitel werden das Vorgehen und die Verwendung von Schema-Elementen für die Schemamodellierung beschrieben und für die Nutzung vom GI4X sinnvoll eingeschränkt. Dies dient der Vereinheitlichung des Schemaentwurfs und der Vermeidung von unnötiger Komplexität.

5.5.2 Allgemeine Festlegungen

In den nachfolgenden Kapiteln werden grundlegende Festlegungen hinsichtlich der Verwendung von XML-Schema beschrieben. Sowohl Absender als auch Empfänger MÜSSEN über geeignete Mechanismen sicherstellen, dass zu den vereinbarten Schemata konforme XML-Daten ausgetauscht werden.

Festlegung: [DARF NICHT] Alle Sprachelemente von XML-Schema, die nachfolgend nicht explizit aufgeführt sind, werden zunächst solange explizit ausgeschlossen, bis begründete Tatbestände für deren Verwendung und damit nachträglicher

expliziten Berücksichtigung sprechen.

Festlegung: [MUSS] Es werden ausschließlich XML in der Version 1.0 und 1.1 sowie XML-Schema in der Version 1.0 unterstützt.

5.5.2.1 Designvorgaben

Für einen Schemaentwurf existieren unterschiedliche Modellierungsmöglichkeiten. Eine Entscheidung zugunsten einer dieser Varianten ist immer vor dem Hintergrund des jeweils betrachteten verfahrensspezifischen Kontexts zu treffen. Insbesondere ergeben sich zwei unterschiedliche Ansätze zur Modellierung von verfahrensspezifischen- und übergreifenden Daten, deren Repräsentation entweder mittels Element- Attribut oder Element- Kindelement Strukturen modellierbar sind.

Festlegung: [SOLL] In aller Regel sind Element – Kindelement Strukturen zu bevorzugen, da sich diese im Bedarfsfall in einer beliebigen Granularität einfacher erweitern lassen. Damit lassen sich die damit verbundenen Änderungsaufwände minimieren.

Festlegung: [KANN] Attribute sollten mit Vorsicht verwendet werden, da diese gerade auch hinsichtlich zukünftiger Änderungen deutlichen Restriktionen unterliegen (insbesondere fehlende strukturelle Granularität).

Festlegung: [SOLL] Für die Modellierung von Meta-Daten, die in aller Regel keinen strukturellen Änderungen unterworfen sind und darüber hinaus eher atomarer Natur sind, kann eine Element- Attribut- Struktur durchaus sinnvoll sein.

Festlegung: [SOLL] Die Definition eines Schemas sollte sich daran orientieren, in welcher Art und Weise die Daten aus fachlicher Sicht strukturiert und nachfolgend weiterverarbeitet werden sollen (vgl. konzeptueller und physischer Schemaentwurf).3

Ausgehend von lokalen und globalen Elementen und Typen sind folgende Designprinzipien zu unterscheiden:

Designprinzip	Art	Beschreibung
Russian Doll	DARF NICHT	Alle Elemente, außer dem Root, und alle Typen sind lokal definiert.
Garden of Eden	DARF NICHT	Alle Elemente und Typen sind global definiert.
Salami Slice	SOLL NICHT	Alle Elemente sind global und alle Typen sind lokal definiert.
Venetian Blinds	SOLL	Alle Elemente sind lokal und alle Typen sind global definiert.

[3] Beim konzeptuellen Schemaentwurf wird eine saubere Abbildung der natürlichen Beziehungen der repräsentierten Realwelt-Objekte in Form spezifischer hierarchischer Strukturen vorgenommen. Der physikalische Schemaentwurf orientiert sich mehr an den Anforderungen der nachgelagerten Prozesse. In diesem Zusammenhang werden häufig flache Strukturen für einen einfacheren Datenbankimport gewählt.

Tabelle 4: Darstellung der XML-Schema-Designprinzipien

Festlegung:	[SOLL] Ein Schemaentwurf sollte auf Grundlage des Venetian Blinds Designprinzip erfolgen, in dem alle Typen global, und Elemente und Attribute grundsätzlich zunächst lokal definiert werden sollten.
Festlegung:	[SOLL] Elemente und Attribute sollten nur dann global definiert werden, wenn diese auch wirklich an anderer Stelle wiederverwendet werden können oder sollen.
Festlegung:	[MUSS] Lokale Definitionen an unterschiedlichen Stellen, jedoch mit gleichem Namen und gleicher inhaltlichen Bedeutung, sind global zu definieren.
Festlegung:	[DARF NICHT] Lokale Definitionen an unterschiedlicher Stelle, jedoch mit gleichem Namen aber unterschiedlicher inhaltlichen Bedeutung, sind unzulässig.

5.5.2.2 Versionierung

Festlegung:	[MUSS] Für die Versionierung sind die Vorgaben aus dem Kapitel 6 – Versionierung – zu berücksichtigen.

5.5.2.3 Root-Element

Festlegung:	[MUSS] Das Root- Element MUSS in der jeweiligen XSD in Form einer geeigneten Kommentierung hervorgehoben werden, da es hierzu keinen durch XML-Schema definierten Mechanismus gibt.
Festlegung:	[SOLL] Jedes Schema sollte lediglich ein Root- Element definieren.

5.5.2.4 Wertelisten

Wertelisten werden dazu verwendet, den Wertebereich und damit die zulässigen Inhalte von Elementen und Attributen sinnvoll einzuschränken. Hierbei ist zwischen statischen und dynamischen Wertelisten zu unterscheiden:

- Statische Wertelisten werden direkt im Schema als Typ definiert, der die zulässigen Werte als Aufzählung vorgibt (vgl. Enumerationen und Listen).
- Dynamische Wertelisten werden in Form von lexikalisch eingeschränkten Typen definiert. Folglich werden die zulässigen Werte nicht direkt im Schema aufgelistet (vgl. Regular Expressions). Vielmehr werden die möglichen Werte in einer zentralen, extern erreichbaren Liste vorgehalten. Der Verarbeitungsprozess MUSS Zugriff auf diese Liste haben.

Das folgende Beispiel definiert einen Datentyp für eine Statische Werteliste vom Typ xs:string mit den Werten „Ja“ und „Nein“:

```
<xs:simpleType name="Antwort_Stp">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Ja"/>
    <xs:enumeration value="Nein"/>
  </xs:restriction>
</xs:simpleType>
```

Das folgende Beispiel definiert eine dynamische Werteliste, die mögliche Verfahrenskennungen durch den regulären Ausdruck "[ET][0-9A-Z]{3}[0-9]" (E oder T gefolgt von exakt 3 Ziffern oder Großbuchstaben und am Ende eine Ziffer) beschreibt:

```
<xs:simpleType name="Verfahrenskennung_Stp">
        <xs:annotation>
            <xs:documentation source="http://.../liste">Liste Kennungen</xs:documentation>
        </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="[ET][0-9A-Z]{3}[0-9]"/>
  </xs:restriction>
</xs:simpleType>
```

Festlegung: [MUSS] Sind die typeinschränkenden Wertelisten endlich und im Zeitablauf eher stabil hinsichtlich der Änderungshäufigkeit, so sind statische Wertelisten zu verwenden. Solche Wertelisten sind als einfache Typen im entsprechenden Schema zu hinterlegen.

Festlegung: [MUSS] Sind die typeinschränkenden Wertelisten im Zeitablauf eher häufigen Änderungen unterzogen und deren Werte nicht abschließend bekannt, so sind dynamische Wertelisten in Form von möglichst exakten lexikalischen Einschränkungen zu verwenden.

Festlegung: [SOLL] Solche Wertelisten sollten einen Verweis auf eine externe Liste enthalten. Hierzu ist <xs:annotation> in Verbindung mit <xs:documentation> zu verwenden. Über das Attribut source SOLL die URL, über die die externe Liste erreichbar ist, angegeben werden.

5.5.2.5 Repräsentation leerer Werte

Manchmal kann es wünschenswert sein, nicht anwendbare oder unbekannte Informationen *explizit* mit einem Element zu repräsentieren, nicht nur durch das Fehlen eines Elements. Es kann zum Beispiel wünschenswert sein, einen “NULL”- Wert, der aus einer relationalen Datenbank kommt, durch ein vorhandenes Element zu repräsentieren. Solche Fälle können durch den Nil- Mechanismus von XML-Schema abgedeckt werden, der es ermöglicht, dass Elemente mit einem echten Wert oder mit einem Nil- Wert auftreten.

Es existieren unterschiedlichste Designmöglichkeiten zur Repräsentation und Unterscheidung von DBNull und leeren Werten mit jeweils unterschiedlichsten Vor – und Nachteilen. Die Festlegung eines einheitlichen Vorgehens ist nicht zuletzt für die Modellierung von **optionalen Feldern** notwendig. Hierfür existieren folgende technische Möglichkeiten:

Modellierungsoption	Art	Beurteilung
Nichterscheinen der jeweiligen Komponente	[SOLL]	Einfacher Mechanismus. Die Komponenten müssen jedoch explizit im Schema als optional deklariert werden, was nicht immer aus fachlicher Sicht gewünscht ist.
Leerer Inhalt der jeweiligen Komponente	[DARF NICHT]	Einfacher Mechanismus. Der Inhalt kann aber genau dann nicht mehr leer sein, wenn der Wertebereich der Komponente entsprechend eingeschränkt wird.
Durch Nutzung des Attributs **<xs:nil>** innerhalb des Instanzdokuments	[KANN]	Zunächst muss hierzu das Attribut **nillable="true"** in **<xs:element>** gesetzt werden. Mit Hilfe dieser Variante können leere Inhalte von nicht vorhandenen Werten unterschieden werden, sofern dies für die jeweilige Komponente aus fachlicher Sicht gewünscht ist. Außerdem sind mit dieser Variante auch leere Inhalte für Komponenten mit Wertebeschränkungen gültig. Dieser Ansatz ist jedoch nicht für Attribute nutzbar.

Tabelle 5: Repräsentation von leeren Werten

Festlegung: [SOLL] Die Darstellung und Unterscheidung von Null- Werten und leeren Werten sollte auf Basis von optionalen Komponenten (Attribute und Elemente) erfolgen. Durch Nichterscheinen der jeweiligen Komponente sind Null- Werte und durch leere Inhalte sind wirklich leere Werte darzustellen. Damit ergibt sich ein einheitlicher Ansatz sowohl für Elemente als auch für Attribute.[4]

5.5.2.6 Gemischtes Inhaltsmodell (Mixed Content)

Zeichen in Mixed Content sind typfreier Inhalt in XML-Schema. Hierbei bezieht sich Mixed Content immer auf das gesamte Inhaltsmodell eines Typs. Diese Inhalte können Zeichen sein, gemischt mit Elementen. Diese Mischung zu beschreiben würde daher über das Modell von Simple Types (die fortlaufende Zeichenketten voraussetzen) hinausgehen.

Festlegung: [DARF NICHT] Mixed Content darf nicht verwendet werden. Es kann in keiner Weise eingeschränkt werden, d.h. dort wo Mixed Content erlaubt ist (Attribut mixed="true"), kann das volle Unicode-Repertoire an Zeichen erscheinen. Dies

[4] vgl. Kapitel 5.5.14.2.4 – Typableitungen (Derivations)

kann ein Problem sein, wenn Applikationen den Zeichenumfang von Instanzen einschränken wollen.

5.5.2.7 Betrachtung von Whitespaces

Festlegung: [MUSS] Bei der Behandlung von Leerzeichen, Tabulatoren u.ä. sind die Vorgaben von [XML-Schema] zu berücksichtigen. Abweichungen und Einschränkungen zu dem beschriebenen Vorgehen sind unzulässig (vgl. hierzu [Whitespace]).
Damit können all diejenigen Wertebereiche von Typen eingeschränkt werden, die von <xs:string> abgeleitet sind. Zulässige Werte für <xs:whiteSpace> sind preserve, replace und collapse.

<xs:whiteSpace> ist auf alle atomaren- und Listen- Datentypen anwendbar. Für alle anderen atomaren Datentypen außer <xs:string> (und Typen, die durch Einschränkung davon abgeleitet sind) ist der Wert von <xs:whiteSpace> collapse und kann vom Schema-Autor nicht geändert werden; für <xs:string> ist der Wert von <xs:whiteSpace> preserve; für jeden Typ, der durch Einschränkung von string abgeleitet ist, kann der Wert von <xs:whiteSpace> einer der drei zulässigen Werte sein. Für alle Datentypen, die von Listen abgeleitet sind, ist der Wert von <xs:whiteSpace> collapse und kann nicht von einem Schema-Autor geändert werden. Für alle Datentypen, die durch Vereinigung abgeleitet sind, findet <xs:whiteSpace> keine direkte Anwendung. Es sei darauf hingewiesen, dass Attribute und Elemente Whitespaces und insbesondere Zeilenenden unterschiedlich behandeln. Der Grund dafür liegt in der Art und Weise, wie der XML-Standard die Normalisierung von Attributwerten selbst definiert. Dieses Verhalten kann aber bei einfachen Typen mit <xs:whiteSpace> gesteuert werden.

5.5.2.8 Platzhalter

Festlegung: Platzhalter (<xs:any> und <xs:anyAttribut>) sind geeignete Mechanismen von XML-Schema, um fest definierte und damit kontrollierbare Bereiche für die Erweiterung innerhalb eines Schemas zur Verfügung zu stellen. Durch die Platzhalter kann Offenheit gegenüber nicht bekannten Inhalten sowie gegenüber vorhersehbaren zukünftigen Erweiterungen erreicht werden. Zudem kann mit dem Attribut processContents="strict" erreicht werden, dass der Parser das darin enthaltene Inhaltsmodell vollständig auf Grundlage des mit dem vorgeschriebenen Namensraum assoziierten Schema validieren muss. Mit processContents="lax" wird der XML-Prozessor angewiesen, das darin enthaltene Inhaltsmodell zu validieren, falls er dazu in der Lage ist. Er wird dann Elemente und Attribute validieren, für die er Schema-Information erlangen kann, aber für diejenigen, wo dies nicht möglich ist, keine Fehler melden.

Es ist folgende Festlegung zu berücksichtigen:
[SOLL NICHT] Platzhalter sollten generell vermieden werden.
[DARF NICHT] Platzhalter dürfen in den Nutzdatenteilen nicht verwendet werden.

[KANN] Platzhalter können auf der Transportebene (vg. Header-Strukturen), wenn diese vorhanden und nutzbar sind, verwendet werden.

5.5.3 Semantik der zulässigen Schemasymbole

Für den grafischen Entwurf von XML-Schemata werden Diagramme und entsprechende Symbole genutzt, die nachfolgend näher erläutert werden sollen. Aus Gründen der Komplexitätsvermeidung und Vereinheitlichung MUSS die Menge an zulässigen Modellierungsmöglichkeiten eingeschränkt werden. [5]

5.5.4 Strukturelemente (Compositors)

5.5.4.1 Darstellung

Elemente auf unterschiedlichen Hierarchieebenen werden mit Hilfe der Strukturelemente verbunden.

Symbol	Beschreibung
	Sequence: Bei diesem Strukturelement wird eine bestimmte Anzahl an Kindelementen in einer festen Reihenfolge ausgewählt.
	Choice: Bei diesem Strukturelement muss genau ein Kindelement ausgewählt werden.

Tabelle 6: Strukturelemente

5.5.4.2 Festlegung

Festlegung: [SOLL] Die Strukturelemente <xs:sequence> und <xs:choice> können verwendet werden.

Festlegung: [DARF NICHT] Das Strukturelement <xs:all> ist nicht zu verwenden, da insbesondere durch dessen Verwendung die Reihenfolge der in dem entsprechenden Inhaltsmodell enthaltenen Elemente beliebig und deren Kardinalität auf {0,1} bzw. {1,1} beschränkt ist.

5.5.5 Kardinalitäten (Cardinality Constraints)

5.5.5.1 Darstellung

Mit Hilfe von Kardinalitäten können Häufigkeitsbeschränkungen für einfache und komplexe Typen sowie für die Strukturelemente festgelegt werden.

Kardinalität	Symbol	Beschreibung
[0..1]		Optionales Element: Es kann keinmal oder genau einmal vorkommen.
[1]		Pflichtelement: Es muss genau einmal vorkommen.

[5] Die grafische Darstellung der Schemasymbole ist dem XMLSpy von der Firma Altova entnommen, einem der Marktführer von XML- basierenden Entwicklungswerkzeugen.

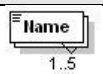
Kardinalität	Symbol	Beschreibung
[n..m]		Wiederholendes Element: Es muss genau n-mal aber darf nur maximal m-mal vorkommen. Im Beispiel muss das Element einmal und darf maximal nur fünfmal vorkommen.

Tabelle 7: Kardinalitäten

5.5.5.2 Festlegung

Festlegung: [SOLL] Die Kardinalitäten unterliegen keinerlei Beschränkungen und sind zu verwenden.

5.5.6 Schemadokumentation (Documentation)

5.5.6.1 Darstellung

XML-Schema ermöglicht die Dokumentation innerhalb einer Schemadatei. Hierzu stehen folgende Sprachmittel zur Verfügung:

```
<xs:annotation>
    <xs:documentation>
        Beschreibung
    </xs:documentation>
    <xs:appinfo>
        Beschreibung: Anwendung / Software
    </xs:appinfo>
</xs:annotation>
```

<xs:documentation> und **<xs:appinfo>** können ein Attribut **source** für eine URL mit weiteren Informationen besitzen.

Weiterhin können entsprechende Detailinformationen in dem Tag **<!-- Dokumentation -->** hinterlegt werden.

5.5.6.2 Festlegung

Festlegung: [MUSS] Lediglich die zuvor beschriebenen Möglichkeiten sind zulässig und somit zu verwenden, um damit neben einer allgemeinen Dokumentation auch insbesondere die Lesbarkeit von Schemadateien zu erhöhen.

Festlegung: [MUSS] Jede einzelne Dokumentation muss kurz und prägnant sein. Längere Erläuterungen sind separat außerhalb des Schemas zu pflegen und über das ´source´ Attribut (Verweis auf das externe Dokument) einzubinden.

Festlegung: [MUSS] Die Versionshistorie muss in der entsprechenden Schemadatei hinterlegt werden. Die protokollierten Änderungen müssen die Versionsnummer,

Änderungsdatum, sowie Grund/Art der Änderung umfassen.

Festlegung: [SOLL] Alle Elemente und Attribute sollten im Schema dokumentiert werden.

Festlegung: [SOLL] Prüfroutinen und sonstige verfahrensspezifische Merkmale, die nicht durch das jeweilige Schema definiert werden, sind in einer eigenständigen Dokumentation außerhalb dieses Schemas zu beschreiben.

Festlegung: [SOLL] xs:documentation enthält menschlich interpretierbare Dokumentation, xs:appinfo maschineninterpretierbare Informationen.

5.5.7 Verwendung von Vereinigungen und Listen (Union and List)

5.5.7.1 Darstellung

Mit Hilfe von **<xs:union>** lassen sich die zulässigen Werte für eine Typdeklaration aus den Werten einer oder mehrerer einfachen Datentypen ableiten.

```
<xs:element name="Dateigroesse_Unn">
  <xs:simpleType>
    <xs:union memberTypes="Union1_Stp Union2_Stp" />
  </xs:simpleType>
</xs:element>

<xs:simpleType name="Union1_Stp">
  <xs:restriction base="xs:positiveInteger">
    <xs:maxInclusive value="100"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="Union2_Stp">
  <xs:restriction base="xs:string">
    <xs:enumeration value="s"/>
    <xs:enumeration value="m"/>
    <xs:enumeration value="l"/>
  </xs:restriction>
</xs:simpleType>
```

Wertelisten mit zulässigen Werten eines bestimmten Datentyps können hingegen mit **<xs:list>** definiert werden.

```
<xs:element name="Name" type="String_Lst">

<xs:simpleType name="String_Lst">
```

```
  <xs:list itemType="xs:string"/>
</xs:simpleType>
```

5.5.7.2 Festlegung

Festlegung: [SOLL] Vereinigungen <xs:union> und Listen <xs:list> sind zu verwenden.

5.5.8 Verwendung von Enumerationen (Enumeration)

5.5.8.1 Darstellung

Mit Hilfe von Enumerationen kann der zulässige Wertebereich von Datentypen mit **<xs:restriction>** eingeschränkt werden. Alternativ kann dies auch mit Hilfe von **<xs:pattern>** über reguläre Ausdrücke ermöglicht werden. Weitere Details zu der Einschränkung von Datentypen können dem Kapitel 5.5.14.1.2 – Wiederverwendung – entnommen werden.

```
<xs:simpleType name="String_Stp">
        <xs:restriction base="xs:string">
            <xs:enumeration value="String1"/>
            <xs:enumeration value=" String2"/>
            <xs:enumeration value=" String3"/>
            <xs:enumeration value=" String4"/>
        </xs:restriction>
    </xs:simpleType>
```

Insbesondere können **<xs:enumeration>** in Verbindung mit Identitätseinschränkungen genutzt werden. Dies wird in dem nachfolgenden Beispiel dargestellt und anschließend im nächsten Kapitel näher beschrieben.

```
    <xsd:simpleType name="Schluessel_VS_Stp">
        <xs:restriction base="xs:string">
            <xs:enumeration value="VS01"/>
            <xs:enumeration value="VS02"/>
            <xs:enumeration value="VS03"/>
            <xs:enumeration value="VS04"/>
            <xs:enumeration value="VS05"/>
            <xs:enumeration value="VS06"/>
            <xs:enumeration value="VS07"/>
            <xs:enumeration value="VS08"/>
            <xs:enumeration value="VS09"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:element name="Ambulante_Fallzahlen">
        <xs:complexType>
```

```
            <xs:sequence>
                <xs:element name="VS_Schluessel" type="Schluessel_VS_Stp"/>
            </xs:sequence>
        </xs:complexType>
        <xs:unique name="Fallzahl_Zaehlweise_Uqe">
            <xs:selector xpath="../Ambulante_Fallzahlen"/>
            <xs:field xpath="VS_Schluessel"/>
        </xs:unique>
    </xs:element>
```

5.5.8.2 Festlegung

Festlegung: [KANN] Enumerationen sind zu verwenden.
Sie sind für die Einschränkung von Typen und Deklarationen und in Verbindung mit Identitätseinschränkungen in solchen Fällen nutzbar, in denen eine Menge an zulässigen Werten (sog. statische Wertelisten) die Grundlage für eine Einschränkung sein soll.

Festlegung: [KANN] Enumerationen können für die Modellierung kleiner, atomarer, sich selten ändernder Schlüsseltabellen eingesetzt werden.

5.5.9 Identitätseinschränkungen und Schlüsseltabellen (Identity Constraints)

5.5.9.1 Darstellung

Mit Hilfe von Identitätseinschränkungen können Eindeutigkeit und Referenzeinschränkungen in Bezug auf die Inhalte eines oder mehrerer Elemente und Attribute formuliert werden.
Eine alternative Modellierung über **<xs:ID>** und **<xs:IDREF>** in einer DTD beschränkt diese Möglichkeiten, da

- die IDs syntaktisch XML-Namen sein müssen,
- die Eindeutigkeit für das gesamte Dokument gelten muss,
- sie nicht geeignet für Elementinhalte ist,
- sie nicht geeignet für Kombinationen aus mehreren Werten ist

und daher nicht zu verwenden ist. [6]

[6] Es lässt sich also nicht ausdrücken, dass eine laufende Nummer kein zweites Mal verwendet werden soll, dass sie nur für diesen Elementtyp eindeutig sein soll, dass sie beispielsweise als Text eines weiteren Kindelements im XML-Dokument angegeben ist oder dass sie nur in Kombination mit dem Familiennamen eindeutig sein soll.

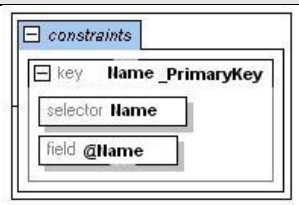
Symbol	Beschreibung
	Identitätseinschränkungen können sowohl durch die Eindeutigkeit **<xs:unique>** als auch durch die Verwendung von Fremdschlüsseln **<xs:key>** und **<xs:keyref>** umgesetzt werden. Folglich ist die XML-Darstellung einer identitätsbeschränkenden Definition für eine Schemakomponente entweder eine Element-Informationseinheit **<key>**, **<keyref>** oder **<unique>**. Identitätseinschränkende Definitionen werden nach dem / den Elementname(n) und dem Zielnamensraum identifiziert, müssen innerhalb eines XML-Schemas eindeutig sein und werden innerhalb eines Bereichs bestimmter Elemente spezifiziert. (vgl. nachfolgendes Beispiel für Fremdschlüsselbeziehungen) [7]

Tabelle 8: Identitätseinschränkungen

Das W3C beschreibt folgende identitätseinschränkende Rollen:

Eindeutigkeit <xs:unique>

```
<xs:unique name="Name_Uqe">
  <xs:selector xpath="Ausgangsmenge" />
  <xs:field xpath="eindeutiger Wert" />
</xs:unique>
```

- <xs:selector> –
  gibt an, innerhalb welcher Menge (von Knoten) bestimmte Werte eindeutig sein sollen.

- <xs:field> –
  gibt an, welcher Wert für jeden Knoten der durch <xs:selector> ausgewählten Menge eindeutig sein soll. Das „xpath“ Attribut enthält einen XPath-Ausdruck, über den die entsprechenden Knoten auswählt werden können. Es kann mehrere <xs:field> -Elemente geben. Diese bilden dann zusammen einen kombinierten Wert.

Fremdschlüsselbeziehungen <xs:key> und <xs:keyref>

```
<xs:element name="ElementName">
  <xs:complexType>
    <!-- das Inhaltsmodell für ElementName -->
  </xs:complexType>
  <xs:key name="Name_Key">
    <xs:selector xpath="Ausgangsmenge"/>
```

[7] Weitere Details sind folgenden URL’s zu entnehmen: W3C – XML Schema, [Constraints1] in Verbindung mit [Constraints2]

```
    <xs:field xpath="eindeutiger Wert " />
  </xs:key>
  <xs:keyref name="Name_Krf" refer="Name_Key">
    <xs:selector xpath="Ausgangsmenge" />
    <xs:field xpath=" eindeutiger Wert " />
  </xs:keyref>
</xs:element>
```

- **<xs:key> –**
  legt einen Primärschlüssel ähnlich wie **<xs:unique>** fest, allerdings dürfen die über **<xs:field>** adressierten Werte nicht fehlen, d.h. sie dürfen sozusagen nicht **nil** sein. [8]

- **<xs:keyref> –**
  definiert einen Fremdschlüssel, wobei der Bezug zum Primärschlüssel angegeben werden muss. Der Wert des Primärschlüssels muss vorhanden sein.

Hinweis:	Es muss darauf hingewiesen werden, dass hier zulässige XPath-Ausdrücke lediglich eine Untermenge der vollen XPath-Sprache darstellen. Außerdem muss darauf geachtet werden, dass eine Selektion von Knoten nicht ins Leere läuft. Hierbei spielen Präfixe, qualifizierte Namen und NCNames eine zentrale und wichtige Rolle.

Somit könnten Schlüsseltabellen direkt in einem Instanzdokument hinterlegt oder extern referenziert werden. Abschließend kann festgehalten werden, dass innerhalb eines XML-Schemas zwar Schlüssel – und Fremdschlüsselbedingungen und damit auch Eindeutigkeiten, nicht jedoch allgemeine Integritätsbedingungen formuliert werden können. [9]

5.5.9.2 Festlegung

Festlegung:	[DARF NICHT] Für die Modellierung von Identitätseinschränkungen dürfen xs:ID/xs:IDREF nicht verwendet werden.
Festlegung:	[SOLL] Für die Modellierung von Identitätseinschränkungen sind <xs:unique>, <xs:key> und <xs:keyref> zulässig und können verwendet werden.
Festlegung:	[MUSS] Die Modellierung von Identitätseinschränkungen ist genau dann verpflichtend, wenn geeignete Schlüssellisten für das Fachverfahren existieren oder die Werte bestimmter Elemente / Attribute eindeutig sein müssen.

[8] vgl. Kapitel 5.5.2.5 – Repräsentation leerer Werte

[9] Beispielsweise können hiermit keine „Wenn …. Dann" – Abhängigkeiten beschrieben werden: Wenn das optionale Element X im Instanzdokument enthalten ist, muss das Element Y den Wert a haben, andernfalls den Wert b.

Festlegung: [DARF NICHT] Öffentliche Schlüsseltabellen dürfen nicht in dem Instanzdokument enthalten sein, sondern müssen vielmehr extern referenziert werden. Zur Gewährleistung der Unveränderbarkeit von öffentlichen Schlüsseltabellen in XML-Instanzdokumenten können Hash- Werte herangezogen werden.

5.5.9.3 Schlüsseltabellen

Schlüsseltabellen dienen dazu, die legalen Werte für einzelne Elemente/Attribute oder für n-Tupel[10] von Elementen/Attributen festzulegen. Die Modellierung von Schlüsseltabellen durch XML-Schema Mittel kann entweder durch Enumerations-Fassetten oder Identitäts-Constraints erfolgen.

Bei der Verwendung von Enumerationen erfolgt die Validierung der Werte durch XML-Schema Validatoren. Eine Prüfung der Werte ist nicht erforderlich, da diese im Schema definiert sind. Enumerationen sind ungeeignet, um n-Tupel Schlüssel zu modellieren.

Bei der Verwendung von Identitätsconstraints müssen die Schlüsseltabellen explizit in den XML-Instanzdokumenten enthalten sein. Die Validierung der Werte erfolgt durch XML-Schema Validatoren. Die Validierungsvorschrift wird über Identitätsconstraints (key/keyref) im XML-Schema hinterlegt. Eine Validierung der Schlüsseltabellendaten ist erforderlich, da die Hinterlegung der Tabelle im Instanzdokument durch den Erzeuger des Dokumentes fehlerbehaftet sein kann. Diese Prüfung kann jedoch nicht mit XML-Schema Mitteln definiert werden, so dass sie durch ein komplementäres Prüfprogramm erfolgen muss. Die Prüfung besteht aus 2 Schritten:

- Überprüfung der Gültigkeit der Tabelle
- Überprüfung der Gleichheit der Tabelle im Dokument gegen die Originaltabelle[11]. MD5 Hashsummen können dabei verwendet werden, um die inhaltliche Gleichheit der Tabelle zu prüfen (Signaturen werden nicht benötigt, können aber eingesetzt werden!).

Die Einbettung der Schlüsseltabelle kann mit diesem Ansatz literal erfolgen (interne Schlüsseltabelle) oder über eine XInclude Referenz (externe Schlüsseltabelle), die durch den Parser aufgelöst wird. Bei der Verwendung von XInclude hat die verarbeitende Instanz eine größere Kontrolle und kann dem Parser gegebenenfalls bereits geprüfte valide Schlüsseltabellen zur Einbettung bereitstellen. Im Gegensatz zu Enumerationen ist eine Validierung von komplexen n-Tupel Schlüsseltabellen mit diesem Ansatz möglich.

Neben der Validierung von Schlüsseltabellen durch XML-Schema Mittel, ist es auch denkbar, dass ein nachgelagertes Prüfprogramm die Validierung der Schlüsselwerte übernimmt.

---

[10] Geordnete Liste von Werten (zusammengesetzte Schlüssel)

[11] Diese muss entweder als Teil der technischen Dokumentation oder über ein öffentliches Repository verfügbar sein.

Die folgenden Festlegungen sollen der Entscheidungsfindung dienen, welcher der 3 Modellierungsansätze für eine Schlüsseltabelle geeignet ist.

Festlegung: [DARF NICHT] Schlüsseltabellen dürfen nicht literal in einem Instanzdokument eingebettet werden, da die dadurch nötigen Prüfungen der Schlüsseltabellendaten die Validierung unnötig komplizieren würde.

Festlegung: [MUSS] Bei der Verwendung von Identitätsconstraints zur Validierung von Werten müssen zugehörige Schlüsseltabellen über XInclude referenziert werden. Der Wert des XInclude href attributes muss eine relative URL sein, die ausschliesslich aus dem Schlüsseltabellen-Dateinamen bestehen darf[12].

Festlegung: [MUSS] Kleine, atomare, statische Schlüsseltabellen, die sich nicht oft ändern, müssen im XML-Schema als Datentyp Enumerations Fassetten modelliert werden.

Festlegung: [DARF NICHT] Schlüsseltabellen die sich oft ändern, dürfen nicht über Datentyp Enumerations Fassetten modelliert werden, da dies zu unnötig vielen Schema-Versionen führen würde.

Festlegung: [KANN] Kleine, sich oft ändernde Schlüsseltabellen als auch mittelgroße Schlüsseltabellen können über Identitätsconstraints/XInclude instanzbasiert modelliert werden.

Festlegung: [KANN] Die Prüfung von Werten gegen Schlüsseltabellen kann durch externe Prüfprogramme erfolgen, soweit keine der obigen MUSS Festlegungen erfüllt ist.

Festlegung:

Das folgende Beispiel dient zur Veranschaulichung, wie die Modellierung von Identitätsconstraints über externe Schlüsseltabellen aussehen könnte. Das Beispiel basiert auf dem fiktiven Verfahren EXXX0 und der zugehörigen fiktiven Schlüsseltabelle EXXX0-PLZ-1.xml, die gültige Postleitzahlen enthält. Das Format der Tabelle ist nicht normativ zu sehen!

```
<Tabelle xmlns="GI4X:/xml-schema/EXXX0-Nutzdaten/1.1">
    ...
    <PLZ>64293</PLZ>
    <PLZ>...</PLZ>
    ...
</Tabelle>
```

Die Nutzdaten enthalten Anschriften mit Postleitzahlen und die über XInclude eingebundene Schlüsseltabelle:

[12] Dies ermöglicht und erzwingt, dass das Verarbeitungsprogramm dem Parser die Datei geeignet übergeben muss.

```
<Nutzdaten
    xmlns      ="GI4X:/xml-schema/EXXX0-Nutzdaten/1.1"
    xmlns:xi ="http://www.w3.org/2001/Xinclude"
    logische_version="1.2.1"
>
 ...
 <Anschrift>
       <PLZ>64293</PLZ>
 </Anschrift>
 ...
    <!—Einbetten der Schlüsseltabelle/>
 <xi:include  href="EXXX0-PLZ-1.xml"/>
</Nutzdaten>
```

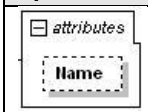
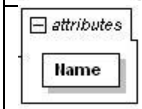
Bei der Definition des Elementes ‚Nutzdaten' müssen nun die Identitätsconstraints noch abgebildet werden:

```
<xs:element name= "EXXX0-Nutzdaten:Nutzdaten"
    xmlns:EXXX0-Nutzdaten="GI4X:/xml-schema/EXXX0-Nutzdaten/1.1">
    ...
    <xs:key name="EXXX0-Nutzdaten:PLZ_Key" >
     <xs:selector xpath=" EXXX0-Nutzdaten:Tabelle/ EXXX0-Nutzdaten:PLZ"/>
     <xs:field xpath="text()"/>
    </xs:key>
    ...
    <xs:keyref name="EXXX0-Nutzdaten:PLZ_Krf" refer="EXXX0-Nutzdaten:PLZ_Key" >
     <xs:selector xpath="EXXX0-Nutzdaten:Anschrift"/>
     <field xpath="EXXX0-Nutzdaten:PLZ"/>
    </xskeyref>
</xs:element>
```

5.5.10 Attribute und Elemente

5.5.10.1 Darstellung

5.5.10.1.1 Attribute

Symbol	Beschreibung
	Optionales Attribut: Es kann keinmal oder genau einmal vorkommen.
	Pflichtattribut: Es muss genau einmal vorkommen.

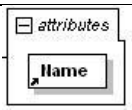
	Referenzattribut: Dieses globale Attribut wurde an einer anderen Stelle im Schema definiert.

Tabelle 9: Attribute

5.5.10.1.2 Elemente

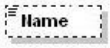
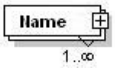
Symbol	Beschreibung
	Optionales Element: Es kann keinmal oder genau einmal vorkommen.
	Pflichtelement: Es muss genau einmal vorkommen.
	Referenzelement: Dieses globale Element wurde an einer anderen Stelle im Schema definiert und muss genau einmal vorkommen.
	Wiederholendes Pflichtelement: Im Beispiel muss es genau einmal und kann maximal fünfmal vorkommen.
	Wiederholendes Element mit Kindelementen: Dieses Element kann ein oder mehrere Kindelemente enthalten. Dieses Inhaltsmodell entspricht somit einem komplexen Typ.

Tabelle 10: Elemente

5.5.10.2 Festlegung

Festlegung: [SOLL] Globale Deklarationen sollten für Elemente / Attribute angewendet werden, die vom Zielschema und von anderen Schemadokumenten wieder verwendet werden. Lokale Elemente / Attribute sollten dann favorisiert werden, wenn die Element- / Attributdeklarationen nur im Zusammenhang mit dem deklarierten Typ sinnvoll sind und nicht für eine erneute Verwendung offen gelegt werden müssen.

Festlegung: [MUSS] Standardmäßig besitzen globale Elemente / Attribute einen Namensraum, der dem Zielnamensraum des Schemas gleicht, während lokale Elemente / Attribute keinen Namensraum besitzen. Da dies die Generierung valider Dokumente unnötig erschwert, müssen alle in einem Schema definierten Elemente den Zielnamensraum des Schemas besitzen und Attribute ohne Namensraum definiert sein. Eine Ausnahme stellen in einem anderen Schema wieder verwendbare, globale Attribute dar, die den Zielnamensraum des Schemas, in dem sie definiert sind, besitzen MÜSSEN.

5.5.11 Gruppen

5.5.11.1 Darstellung

5.5.11.1.1 Attributgruppen

Durch die Verwendung von Attributgruppen kann die Lesbarkeit eines Schemas verbessert werden und außerdem wird damit die Wartung erleichtert, da eine Attributgruppe nur an einer Stelle definiert und gewartet werden muss, aber in vielen Definitionen und Deklarationen referenziert werden kann.

Aus diesem Grund stellen Attributgruppen eine ähnliche Möglichkeit zur Modularisierung und Wiederverwendung bereit wie globale Typdeklarationen.

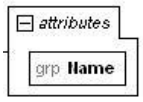
Symbol	Beschreibung
	Attributgruppen stellen eine Gruppierung von mehreren Attributdeklarationen dar – ein modular gebündelter Satz von Attributen. Die einzelnen Attributdeklarationen können dabei sowohl lokal als auch Verweise auf globale Attributdeklarationen sein.

Tabelle 11: Attributgruppen

5.5.11.1.2 Modellgruppen

Mit Hilfe von Modellgruppen können Gruppen von Elementdeklarationen zusammengefasst und wieder verwendet werden. Sie sind dabei jedoch kein Ersatz für komplexe Typen, da sie weder als Attributdeklarationen noch als Typ einer Elementdeklaration spezifiziert werden können.

Symbol	Beschreibung
	Durch Modellgruppen können mehrere Elementdeklarationen zusammengefasst und diese für komplexe Typen als Gruppe eingebunden werden.

Tabelle 12: Modellgruppen

5.5.11.2 Festlegung

Festlegung: [SOLL] Attribut- und Modellgruppen sind zu verwenden. Sie tragen zur Modularisierung und Wiederverwendung bei. Modellgruppen können jedoch keine komplexen Typen ersetzen.

5.5.12 Einfache Typen (Simple Types)

5.5.12.1 Darstellung

Einfache Typen dienen der Beschreibung von Daten (Attributinhalte, Elementinhalte ohne Kindelemente) und dürfen somit selbst weder Kindelemente noch Attribute enthalten.

```
<xs:element name="Test" type="Test_Stp"/>
<xs:simpleType name="URIencodableValue_Stp">
        <xs:restriction base="xs:anyURI"/>
</xs:simpleType>
<xs:simpleType name="Test_Stp">
        <xs:restriction base="URIencodableValue_Stp">
            <xs:enumeration value="http://www.test.de/1"/>
            <xs:enumeration value="http://www.test.de/2"/>
        </xs:restriction>
```

```
</xs:simpleType>
```

5.5.12.2 Festlegung

Festlegung: [MUSS] Die Verwendung der integrierten Datentypen von XML-Schema ermöglicht eine Beschränkung der Werte von Elementen und Attributen als Zeichenfolgen, Daten oder numerische Daten. Damit können die Inhalte von XML-Daten auf interoperable und plattformunabhängige Art und Weise geprüft werden. Aus Praktikabilitätsgründen ist die Nutzung auf die folgenden vordefinierten Primitivtypen von XML-Schema zu beschränken:

string, normalizedString, token, boolean base64binary, hexBinary, float, decimal, integer, long, int, double, anyURI, QName, duration, dateTime, time, date, gYearMonth, gYear, gMonthDay, gDay, gMonth.

Festlegung: [SOLL] Bei der Verwendung der obigen Datentypen soll der spezifischste Datentyp verwendet werden. So soll zum Beispiel int – und nicht Integer – verwendet werden, wenn der Wertebereich von int zur Modellierung der Werte ausreicht.

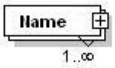
Festlegung: [KANN] Weiterhin ist der zulässige Wertebereich mit Hilfe von regulären Ausdrücken und / oder Enumerationen und Identitätseinschränkungen sinnvoll einzugrenzen.
Insbesondere ist der abgeleitete Datentyp <xs:token> dem primitiven Datentyp <xs:string> für die Definition von reinen „Textfeldern“ vorzuziehen.

Festlegung: [KANN] Eine Einschränkung und Erweiterung von einfachen Typen ist zulässig. [13] Insbesondere sind Ableitungen mittels Fassetten zu verwenden.

5.5.13 Komplexe Typen (Complex Types)

5.5.13.1 Darstellung

Komplexe Typen ergeben sich entweder durch eine Erweiterung eines einfachen Typen oder durch die Definition eines Inhaltsmodells bestehend aus einfachen und komplexen Typen, die mittels der Strukturelemente verbunden werden.

Symbol	Beschreibung
Globaler komplexer Typ  *Lokaler* komplexer Typ 	Komplexer Typ: Dieser Typ enthält ein Inhaltsmodell, welches sich aus Kindelementen und / oder Attributen zusammensetzt, die mit den zuvor beschriebenen Strukturelementen verknüpft sind. Das sich daraus ergebende Inhaltsmodell legt die Sub-struktur fest. Der Unterschied zwischen lokalen – und globalen

[13] Ein Beispiel für eine Erweiterung wäre eine Situation, in der eine Elementdeklaration einen einfachen Typ als Inhalt besitzt und über mindes-tens ein Attribut verfügt.

	Deklarationen wird in dem nachfolgenden Kapitel 5.5.14.1.1 – Globale Elemente und Attribute – beschrieben.

Tabelle 13: Komplexe Typen

5.5.13.2 Festlegung

Festlegung: [MUSS] Komplexe Typen sind zu verwenden. Sie tragen zur Modularisierung und Wiederverwendung bei.

Festlegung: [KANN] Eine Einschränkung und Erweiterung von komplexen Typen ist zulässig. Jedoch sind Einschränkungen mit Vorsicht zu verwenden, da sie mitunter sehr komplex werden können und im Gegensatz zur Erweiterung weder den Konzepten der objektorientierten Programmierung noch denen der relationalen Datenbanken genau entsprechen. Insbesondere muss darauf geachtet werden, dass jede Änderung an einem Vorgängertyp manuell in die gesamte Ableitungsstruktur übertragen werden muss. Sie können jedoch durchaus zweckmäßig sein, wenn sekundäre Typen einem generischen primären Typen entsprechen müssen, aber dennoch ihre eigenen Einschränkungen verwenden, um über die des primären Typs hinauszugehen.14

5.5.14 Modularisierung und Wiederverwendung beim Schemaentwurf

5.5.14.1 Darstellung

5.5.14.1.1 Globale Elemente und Attribute

Globale Elemente und Attribute (benannte Typen) werden durch Deklarationen erzeugt, die als Kind des **schema**- Elements auftreten. Nach ihrer Deklaration können globale Elemente und Attribute in einer oder mehreren Deklaration(en) über das Attribut **ref** referenziert werden. Eine Deklaration, die ein globales Element referenziert, bewirkt, dass das referenzierte Element im Instanzdokument im Kontext der referenzierenden Deklaration auftreten darf.

Es gibt eine Reihe von Einschränkungen bei der Verwendung von globalen Elementen und Attributen, auf die das W3C entsprechend hinweist. Eine ist, dass globale Deklarationen keine Referenzen enthalten dürfen, sie müssen einfache wie komplexe Typen direkt benennen. Genauer gesagt, globale Deklarationen dürfen das Attribut **ref** nicht enthalten, sie müssen das Attribut **type** verwenden (oder die Definition eines anonymen Typs enthalten). Eine zweite Einschränkung ist, dass die Kardinalität globaler Elemente und Attribute nicht beschränkt werden kann, wohl aber die Kardinalität lokaler Elemente/Attribute, die eine globale Deklaration referenzieren. Mit anderen Worten, globale Deklarationen dürfen die Attribute **minOccurs**, **maxOccurs** und **use** nicht enthalten.

14 Weitere Details sind dem Kapitel 5.5.14.1.2 – Wiederverwendung – zu entnehmen.

Ein lokaler (anonymer) Datentyp ist hingegen ein Typ, der in der Definition eines Elements geschachtelt ist und nicht auf einen globalen Typ verweist. Somit kann auf diesen Typ nicht referenziert werden.

5.5.14.1.2 Wiederverwendung

Neben globalen Elementdefinitionen, die mehrfach benutzt werden können, lassen sich auch sog. globale Typen definieren, die für verschiedene Elemente wiederverwendet werden können.
Ähnlich wie bei den objektorientierten Programmiersprachen „erben" die abgeleiteten Typen die Merkmale ihrer Vorgänger. Die Ableitung neuer Typen kann nur auf Basis benannter Typen erfolgen. Eine Ableitung erfolgt mit Hilfe des Attributs **base**.

XML-Schema ermöglicht folgende Ableitungsvarianten:

- Ableitung durch Erweiterung **<xs:extension>**
  Der Ausgangstyp wird um zusätzliche Merkmale erweitert, während der bestehende Teil der Typdefinition unverändert bleibt. [15] Einfache Ausgangstypen behalten ihren einfachen Inhalt und können ausschließlich um Attribute erweitert werden.

```
<xs:element name="Betrag">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:decimal">
        <xs:attribute name="waehrung" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

  Bei komplexen Typen kann dagegen neben einer Aufnahme zusätzlicher Attribute auch das Inhaltsmodell erweitert werden. Die entstehenden Typen haben somit insgesamt stets komplexen Charakter. Durch das Attribut **base** wird der Ausgangstyp für die Ableitung referenziert.

- Ableitung durch Einschränkung **<xs:restriction>**
  Ausgangspunkt dieser Ableitungsvarianten können sowohl einfache als auch komplexe Typen sein:

    - **Komplexe Typen**
      Hier kann die gesamte Typdefinition eingeschränkt werden. Hierzu zählt neben dem „Entfernen" von Komponenten auch eine Verschärfung der Häufigkeitsbeschränkun-

[15] Diese Merkmale werden an das bestehende Inhaltsmodell angehängt.

gen von Elementen und Inhaltsmodellen. Außerdem können optionale Attribute zu Pflichtattributen geändert oder verboten werden.

- **Einfache Typen**
  Hierdurch kann die Menge zulässiger Zeichenfolgen entsprechend beschränkt werden.

  XML-Schema stellt entsprechende Sprachmittel für die Deklaration eigener Datentypen (Ableitung mittels Fassetten) bereit. Abhängig vom Basistyp kann es folgende Fassetten geben:

  - **<xs:length>**
  - **<xs:minLength>**
  - **<xs:maxLength>**
  - **<xs:minInclusive>**
  - **<xs:minExclusive>**
  - **<xs:maxInclusive>**
  - **<xs:maxExclusive>**
  - **<xs:totalDigits>**
  - **<xs:fractionDigits>**
  - **<xs:enumeration>**
  - **<xs:pattern>** – Verwendung regulärer Ausdrücke [RegExpr]

  Jede Fassette besitzt ein Attribut **value**.

In den nachfolgenden Abbildungen werden die Möglichkeiten durch Ableitung beispielhaft dargestellt.

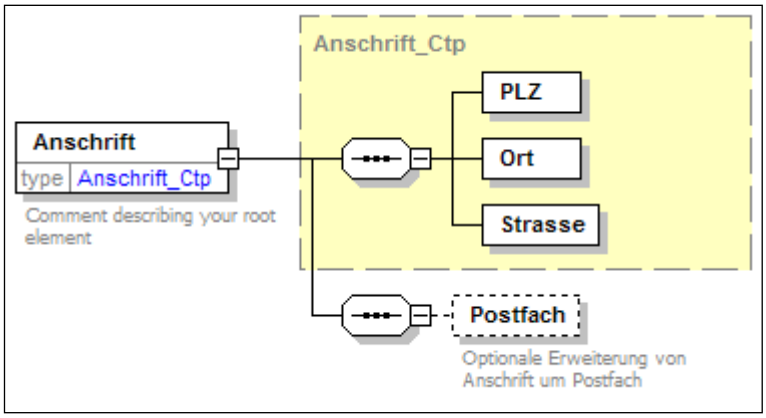


Abbildung 2: Beispiel – Ableitung durch Erweiterung

```
<xs:element name="Anschrift">
  <xs:complexType>
```

```
            <xs:complexContent>
                <xs:extension base="Anschrift_Ctp">
                    <xs:sequence>
                        <xs:element name="Postfach" minOccurs="0">
                         <xs:annotation>
                              <xs:documentation>Optionale Erweiterung von Anschrift um Post-
fach</xs:documentation>
                         </xs:annotation>
                        </xs:element>
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="Anschrift_Ctp">
        <xs:sequence>
            <xs:element name="PLZ"/>
            <xs:element name="Ort"/>
            <xs:element name="Strasse"/>
        </xs:sequence>
    </xs:complexType>
```

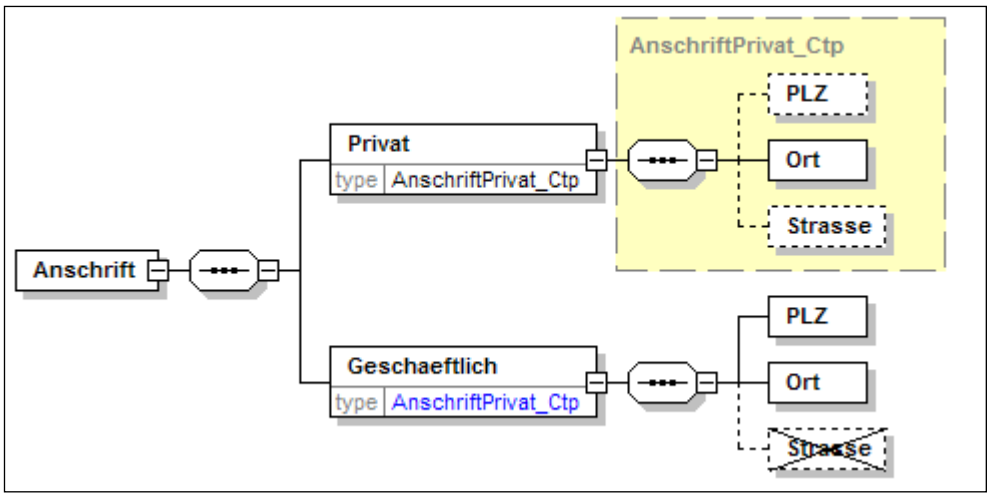


Abbildung 3: Beispiel – Ableitung durch Einschränkung

```
<xs:complexType name="AnschriftPrivat_Ctp">
        <xs:sequence>
            <xs:element name="PLZ" minOccurs="0"/>
            <xs:element name="Ort"/>
            <xs:element name="Strasse" minOccurs="0"/>
        </xs:sequence>
</xs:complexType>
```

```
<xs:element name="Anschrift">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Privat" type="AnschriftPrivat_Ctp"/>
                <xs:element name="Geschaeftlich">
                    <xs:complexType>
                        <xs:complexContent>
                        <xs:restriction base="AnschriftPrivat_Ctp">
                        <xs:sequence>
                            <xs:element name="PLZ"/>
                            <xs:element name="Ort"/>
                            <xs:element name="Strasse" minOccurs="0" maxOccurs="0"/>
                        </xs:sequence>
                        </xs:restriction>
                    </xs:complexContent>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Die Ableitung einzelner Typen kann verhindert werden, indem dem entsprechenden **<xs:complexType>** bzw. **<xs:simpleType>** Element das Attribut **final** hinzugefügt wird. Analog dazu kann in der Typdefinition eines Elementes über das Attribut **block** die Verwendung abgeleiteter Typen verboten werden. Neben diesen granularen Beschränkungsmöglichkeiten besteht die Möglichkeit mit Hilfe des Attributs **blockDefault** bzw. **finalDefault** von **<xs:schema>** diese Beschränkungsmöglichkeiten von **block** bzw. **final** global für das betreffende Schema zu setzen. [16]

5.5.14.1.3 Namensräume, Präfixe und Abhängigkeiten

Ein Schema kann als eine Sammlung (ein Vokabular) von Typdefinitionen und Deklarationen von Elementen angesehen werden, deren Namen zu einem bestimmten Namensraum gehören, dem sog. Ziel-Namensraum. Mit Hilfe von Ziel-Namensräumen kann zwischen Definitionen und Deklarationen aus verschiedenen Vokabularen unterschieden werden.

Um zu überprüfen, ob ein Instanzdokument einem oder mehreren Schemata entspricht, muss eine Zuordnung der Definitionen und Deklarationen von Attributen und Elementen zu den im Dokument

[16] Zulässige Werte sind: (*#all* | Liste aus (*extension* | *restriction*)). Im Zuge der Verwendung von Ersetzungsgruppen kann das **block** Attribut zusätzlich den Wert substitution enthalten. Nähere Details zu den Ersetzungsgruppen sind dem Kapitel 5.5.14.2.5 – Typersetzungen (Substitutions) – zu entnehmen.

benutzten Attributen und Elementen gefunden werden. Die Ziel- Namensräume spielen bei dieser Zuordnung die Hauptrolle.

Jeder Namensraum ist durch einen Uniform Resource Identifier (URI) eindeutig beschrieben und dient insbesondere auch der Vermeidung von Namenskonflikten. Verwendet nun ein XML-Dokument Elemente und Attribute aus verschiedenen Namensräumen, so können diese durch ein sog. Präfix vor jedem Element und Attribut eindeutig unterschieden werden können.

Definition: **XML-Namensräume** bieten eine einfache Möglichkeit, um Element- und Attributnamen, die in XML-Dokumenten verwendet werden können, eindeutig zu benennen. Die Element- und Attributnamen werden mit Namensräumen verknüpft, die durch URI- Verweise identifiziert werden. Damit können unterschiedliche Vokabulare innerhalb eines Instanzdokuments genutzt sowie eindeutig identifiziert und kombiniert werden.

Der allgemeine Aufbau eines **URI-Verweises** wird in [RFC 2396] spezifiziert und stellt entweder eine URL (Uniform Ressource Locator) oder ein URN (Uniform Ressource Name) dar. Dabei werden URL's verwendet, um die Position von Ressourcen im Internet anzugeben, während URN's dauerhafte, positionsunabhängige Bezeichner für Informationsressourcen darstellen.

Ein (bei Elementen optionaler) **Präfix** dient der Kennzeichnung der Zugehörigkeit von Elementen und Attributen zu einem bestimmten Namensraum.

Die Definition eines komplexen Namensraums kann so umfangreich werden, dass die Möglichkeit zur Modularisierung, d. h. dass Aufteilen in mehrere Teilschemata, sinnvoll werden kann. Mit Hilfe von <**xs:include**> kann diese Verteilung wieder im Hauptschema zusammengeführt werden.

Hinweis: Nur Schemata des gleichen Namensraums können <**xs:include**> nutzen.

Um innerhalb eines Schemas auf die Definitionen aus anderen Namensräumen referenzieren zu können, bedarf es der Anweisung <**xs:import**>.

Mit Hilfe der <**xs:redefine**> Methode können einzelne Typen eines Schemas, das zum selben Namensraum gehört, umdefiniert werden.

Es darf dabei keine komplette Neudefinition stattfinden, es müssen vielmehr Ableitungen durch Einschränkung oder Erweiterung, so wie sie im Kapitel 5.5.14.1.2 – Wiederverwendung – beschrieben wurden, verwendet werden. Der Ausgangstyp der Ableitung muss hierbei immer identisch mit dem Namen des erneut zu definierenden Typs sein, der redefiniert wird. Weitere Details sind dem Kapitel 5.5.14.2.6 – Typen – und Gruppenneudefinitionen – zu entnehmen.

5.5.14.2 Festlegung

Zusätzlich zu den bereits getroffenen Festlegungen sind für die Modularisierung und Wiederverwendung die nachfolgenden Festlegungen zu berücksichtigen.

5.5.14.2.1 Wiederverwendung

Festlegung: [MUSS] Lokale Typen und insbesondere Elemente und Attribute müssen immer dann global deklariert bzw. definiert werden, wenn diese an unterschiedlichsten Stellen mehrfach verwendet werden. Damit kann der Grad an Modularisierbarkeit und Wiederverwendbarkeit erhöht werden.

5.5.14.2.2 Namensräume und Präfixe

Festlegung: [MUSS] Die Verwendung von Namensräumen ist für Elemente verbindlich (vgl. Kapitel 5.2 Namensraumhierarchie und 5.5.10.2 Attribute und Elemente). Für jeden importierten Namensraum MUSS ein Präfix im <**xs:schema**> Element deklariert werden. Dies hat u. a. Auswirkung auf

- Verweise auf globale Elemente, Attribute oder Typdeklarationen
- Verwendung von XPath-Ausdrücken für Identitätseinschränkungen.

Festlegung: Der Aufbau wird in der Tabelle 2: Darstellung der Namensraumhierarchie geregelt und MUSS den Vorgaben in [RFC 2396] genügen.

5.5.14.2.3 Namensraumqualifizierung

Das <**xs:schema**>-Element besitzt die Attribute **elementFormDefault** und **attributeFormDefault**, die festlegen, ob lokale Deklarationen im Schema innerhalb des Zielnamensraums qualifiziert werden können oder nicht. Mit der Qualifizierung werden Elemente und Attribute direkt an einen bestimmten Namensraum gebunden.

Festlegung: [MUSS] Der Wert des „elementFormDefault“ Attributes muss “qualified“[17] sein. Allen Elementdefinitionen (auch lokalen) wird durch diese Einstellung ermöglicht, einen Namensraum zu besitzen. Dieser ist (soweit keine Präfixe in Namen verwendet werden) der Zielnamensraum des Schemas.

Festlegung: [MUSS] Der Wert des „attributeFormDefault“ Attributes muss “unqualified“ sein. Lediglich globale Attribute können durch einen Namensraum qualifiziert werden. Anders als Elemente müssen Attribute, die qualifiziert werden müssen, ein Präfix tragen, weil Namensräume keinen voreingestellten Namensraum (Zielnamensraum) für Attribute vorsehen. Attribute, die nicht qualifiziert sein müssen, erscheinen im Dokument ohne Präfix, was der Normalfall ist.

[17] Die Verwendung des Wertes “unqualified” führt dazu, dass nur global definierte Elemente einen Namensraum besitzen können. Dies bedeutet, dass die Festlegung in 5.5.10.2 nicht erfüllbar wäre!

5.5.14.2.4 Typableitungen (Derivations)

Festlegung: [KANN] Typableitungen sind in Form von Einschränkungen und Erweiterungen in der zuvor beschriebenen Art und Weise zulässig und zu verwenden.

Festlegung: [MUSS] Für diejenigen Typdefinitionen, die nicht weiter abgeleitet bzw. keine abgeleiteten Typen verwenden dürfen, sind die Attribute final bzw. block entsprechend zu setzen.

Festlegung: [SOLL] Eine Einschränkung des Wertebereichs in Form von statischen oder dynamischen Wertelisten ist vorzunehmen, soweit die Kriterien hierzu bekannt und vom Fachverfahren vorgegeben werden können.

Festlegung: [SOLL NICHT] Einschränkungen von Wertebereichen mittels <xs:length> sind zu vermeiden.

Festlegung: [SOLL] Einschränkungen von Wertebereichen mittels <xs:minLength> und <xs:maxLength> sind zu verwenden.

Festlegung: [MUSS] Alle Pflichtelemente und Attribute müssen eine Mindestlänge von 1 haben.

Festlegung: [KANN] Alle optionalen Elemente und Attribute können eine Mindestlänge von 0 haben. Dies ist insbesondere für die Repräsentation von leeren Werten notwendig (vgl. Kapitel 5.5.2.5 – Repräsentation leerer Werte).

5.5.14.2.5 Typersetzungen (Substitutions)

Festlegung: [DARF NICHT] Typersetzungen mittels Ersetzungsgruppen oder xs:type dürfen nicht verwendet werden. Hierzu MUSS auch das Attribut blockDefault="substitution" von <xs:schema> gesetzt werden.

Mit Hilfe von Ersetzungsgruppen können ein oder mehrere Elemente für ein globales Element, dem sog. Head-Element, als ersetzbar gekennzeichnet werden. Dadurch kann alternativ das Head-Element durch eines dieser Elemente im Inhaltsmodell ersetzt werden, wodurch die Flexibilität und Erweiterbarkeit erhöht werden kann. Restriktionen sind zwar mit den Attributen block und final möglich. Trotzdem wird es schwierig, auf Basis solcher Schemata gültige Instanzdokumente zu erstellen.

Substitution in XML-Schema kann einerseits über das xs:type Attribut, andererseits über den Mechanismus der Substitution Group erfolgen. Bei xs:type wird dies separat über ein Attribut direkt im Instanzdokument signalisiert. In beiden Fällen werden die Folgeverarbeitung, das Verständnis und die Arbeit mit den Instanzdokumenten erschwert.

5.5.14.2.6 Typen – und Gruppenneudefinitionen (Redefines)

Festlegung: [DARF NICHT] Neudefinitionen durch <xs:redefine> sind nicht zu verwenden. Die Anwendung von Neudefinitionen wirkt sich nicht nur auf das einbettende Schema, sondern auch auf das eingebettete Schema aus. Somit zeigen alle Verweise auf den ursprünglichen Typ dann auf den neudefinierten Typ – der alte Typ bleibt verdeckt. Dies kann zu Problemen und Konflikten führen, da dies im Gegensatz zu einer Ableitung nicht durch die Verwendung der block oder final Attribute verhindert bzw. begrenzt werden kann. Damit kann die ursprüngliche Semantik vollständig geändert werden (Gefahr des Wildwuchses).

5.5.14.2.7 Abstrakte Typen (Abstract Types)

Festlegung: [SOLL NICHT] Abstrakte Typen sind mit Vorsicht zu verwenden, da zwar deren Anwendung nicht so schwierig ist, aber die damit implizit verbundenen Auswirkungen mitunter sehr schwerwiegend und komplex sein können. Abstrakte komplexe Typen und Elementdeklarationen werden häufig für die Erstellung von generischen Basistypen verwendet, die gemeinsame Informationen für einen Satz von Typen enthalten und deren Definition jedoch noch unvollständig sind und durch Ableitungen konkretisiert werden müssen (Polymorphismus).

Eine solche Elementdeklaration kann nicht verwendet werden, um ein Element in einem Instanzdokument zu prüfen. Sie kann außerdem in Inhaltsmodellen nur über Ersetzung auftreten. Genauso kann eine abstrakte komplexe Typdefinition nicht verwendet werden, um ein Element in einem Instanzdokument zu prüfen. Eine abstrakte komplexe Typdefinition kann aber als abstraktes, übergeordnetes Element eines abgeleiteten Typs eines Elements verwendet werden, oder in Fällen, in denen der Typ des Elements in der Instanz durch die Verwendung von xs:type übergangen wird (Typersetzung). Obwohl es nicht notwendig ist, abstrakte Typen in Verbindung mit xs:type zu verwenden, bieten sie einige Vorteile in Situationen, in denen ein generisches Format erstellt wird, für das domänenspezifische Erweiterungen mit einer großen Wahrscheinlichkeit erstellt werden dürfen.

5.5.14.2.8 Chamäleon-Schemata

Festlegung: [DARF NICHT] Chamäleon-Schemata dürfen nicht verwendet werden. Chamäleon-Schemata besitzen keinen eigenen Zielnamensraum, sodass normalerweise nur Elemente und Attribute ohne einen Namensraumnamen geprüft werden können. 18 Wenn nun ein solches Schema innerhalb eines anderen Schemas inkludiert wird, setzt das Schema ohne Zielnamensraum den Zielnamensraum des einbettenden Schemas voraus. Daraus können nicht unwesentliche Probleme entstehen. Beispielsweise sollten Chamäleon- Schemata

[18] Das Inkludieren eines Schemas mit anonymem Zielnamensraum wird als Chamäleon-Schema bezeichnet, da dieses Schema den Namensraum des einbindenden Schemas annimmt.

nicht in Verbindung mit Identitätseinschränkungen (vgl. Kapitel 5.5.9- Identitätseinschränkungen und Schlüsseltabellen (Identity Constraints)) verwendet werden, da lediglich die QName-Verweise auf Typdefinitionen und Deklarationen im Chamäleon- Schema in den Namensraum des einbettenden Schema gezwungen werden können, nicht jedoch die XPath-Ausdrücke der Identitätseinschränkungen.

6 Versionierung

6.1 Allgemeine Anforderungen und Festlegungen

GI4X und die davon abgeleiteten verfahrensspezifischen Schnittstellen können, aus fachlichen, organisatorischen und / oder technischen Gründen, unterschiedlichsten Änderungen und Anpassungen im Zeitablauf unterworfen sein. Daraus ergibt sich der Bedarf für eine einheitliche Konvention zur eindeutigen Bezeichnung von Schnittstellenversionen. In diesem Zusammenhang müssen auch evtl. mögliche Abwärtskompatibilitäten über Versionsbezeichner darstellbar sein.

Durch die Vergabe von einheitlichen Versionsnummern sollen insbesondere die Schemadateien zeitlich parallel nutzbar und referenzierbar sein. Weiterhin kann damit deren Verwendungszweck (externe und interne Nutzung) gekennzeichnet werden.

Die Versionierung von Schnittstellenversionen und Schemadateien bildet somit die Grundlage für die Erweiterbarkeit und Änderbarkeit von Schnittstellen. Darüber hinaus soll sie die nötigen Voraussetzungen für einen möglichen Mehrversionsbetrieb schaffen, das heißt die parallele Nutzung von Schnittstellenversionen ermöglichen.

6.2 Allgemeiner Aufbau und Vergabe von Versionsnummern

XML-Empfehlungs kompatible Versionsnummern orientieren sich an den Vorgaben der [gematik1] und besitzen folgenden Aufbau:

Bestandteil	Eigenschaft	Art	Beschreibung
Hauptversionsnummer (HVNR)	Numerisch, maximal 3 Stellen	MUSS	Die HVNR SOLL sich erhöhen, falls signifikante Änderungen durchgeführt werden, die zur aktuellen Version inkompatibel sind.
Nebenversionsnummer (NVNR)	Numerisch, maximal 3 Stellen	MUSS	Die NVNR SOLL sich erhöhen, falls Erweiterungen an der Schemadatei vorgenommen werden, die kompatibel zu den Schemadateien mit der gleichen HVNR sind.
Revisionsnummer (RENR)	Numerisch, maximal 3 Stellen	MUSS	Die RENR MUSS sich erhöhen, wenn die Änderungen für die Schemavalidierung irrelevant sind.

Tabelle 14: Versionierung – Allgemeiner Aufbau von Versionsnummern in Schemadateien

Festlegung: Die Schemaversion MUSS in dem Attribut version von <xs:schema> hinterlegt werden. Neben der Schemaversion MUSS auch die HVNR und NVNR in dem Namensraum in Form einer Namensraumversion hinterlegt werden (vgl. hierzu Kapitel 5.2 – Namensraumhierarchie). Die Namensraumversion ist zweiteilig, besteht maximal aus 7 Stellen und besitzt den allgemeinen Aufbau

[HVNR].[NVNR].

Zur Kennzeichnung von strukturellen und semantischen Änderungen in verfahrensspezifischen Schemadateien MUSS die sog. logische Version verwendet werden, die in ihrem Aufbau der Schemaversion entspricht.

Die erste produktive Schemaversion sollte mit 1.0.0 beginnen.
Führende Nullen, wie z.B. 01.0.0 sind nicht erlaubt.

Nicht jede fachliche Änderung bedingt eine entsprechende Änderung an der Schemadatei. Hierzu zählen beispielsweise Änderungen an der Semantik bestimmter Inhalte, die zwar nicht zu Änderungen an der zugrunde liegenden Schemadatei führen, jedoch bei der Interpretation dieser fachlichen Inhalte zu berücksichtigen sind. Hierzu ist die logische Version zu verwenden. Folglich muss eine Änderung der logischen Version nicht zwangsweise eine Änderung der Schemaversion bedeuten. Andererseits hat jede Änderung der HVNR oder NVNR der Schemaversion eine Änderung der logischen Version zur Folge.

Festlegung: Verfahrensspezifische Nachrichten MÜSSEN das Attribut logische_version in ihrem Wurzel- Element enthalten19. Die korrespondierende verfahrensspezifische Schemadatei MUSS entsprechend erweitert werden.

Festlegung: Aufeinander folgende Versionsummern MÜSSEN aufsteigend vergeben werden, so dass für eine Versionsnummer vn2 die auf vn1 folgt immer gilt:

a) HVNR(vn2) > HVNR(vn1), oder
b) HVNR(vn2) = HVNR(vn1) UND NVNR(vn2) > NVNR(vn1) , oder
c) HVNR(vn2) = HVNR(vn1) UND NVNR(vn2) = NVNR(vn1) UND RENR(vn2) > RENR(vn1)

Legale Beispiele für aufeinander folgende Versionsnummern sind:

2.0.0 > 1.2.2, da a) erfüllt ist; 2.3.0 > 2.2.4, da b) erfüllt ist; 2.3.4 > 2.3.3, da c) erfüllt ist. Wohingegen 3.3.100 < 3.4.0 gilt, da weder a), b) noch c) erfüllt sind.

Festlegung: Aufeinander folgende Versionsnummern SOLLEN, sofern sich Änderungen in einer Komponente ergeben, in Einser-Schritten erhöht werden.

Die logischen Versionen sind zu den Schemaversionen der korrespondierenden Schemadateien unterschiedlich, jedoch sind diese eindeutig über eine Zuordnungstabelle zuzuordnen.
Die Zuordnungstabelle MUSS zentral verwaltet und öffentlich bereitgestellt werden.

[19] Die Wurzel einer Nachricht kann mit der Wurzel des Instanzdokumentes zusammenfallen. Wenn Nachrichten aber in ein Transportformat eingebettet werden, trifft dies nicht mehr zu, siehe auch das Beispiel in Kapitel 7.2.

6.3 Schemadatei Versionierung

6.3.1 Änderungsaspekte

Die folgende Liste gibt einen Überblick über mögliche Änderungen an Schema-Dateien:

- Initiale Erstellung einer XML-Schema-Datei

- Änderungen an referenzierten Schema-Dateien
    - Das Schema wird dahingehend geändert, dass eine neue Version eines anderen Schemas importiert oder inkludiert wird.
    - Das Schema wird in mehrere Komponenten-Schemas aufgeteilt oder aber umgekehrt Komponenten-Schemas werden durch literales Einfügen aufgelöst.

- Änderungen an Datenstrukturen
    - Hinzufügen/Entfernen von Attributen und Elementen
    - Hinzufügen/Entfernen von Datentyp-Definitionen (Simple-Types, Complex-Types)
    - Ändern der Reihenfolge (Sequenz/Alternative)
    - Änderung der Kardinalität (Optionales und/oder mehrfaches Auftreten von Attributen/Elementen)

- Änderungen an Datentypen von Attributen und/oder Elementen
    - Redefinition des Datentyps (xs:Integer → xs:string)
    - Ersetzung mehrerer gleicher lokaler Typ-Definitionen durch einen global definierten Typ

- Änderung des Wertebereichs (Fassetten) eines Typs
    - Ein Attribut wurde initial als xs:string definiert, danach bekommt es bei einer Änderung einen einschränkenden regulären Ausdruck zugewiesen.
    - Ein Element vom Typ xs:int bekommt einen min/max value zugewiesen.
    - Der Enumerationstyp eines Attributs/Elementes wird durch Hinzufügen und/oder Wegnehmen von Einträgen geändert.

- Umgruppieren und Kommentieren
    - Die Schema-Datei wird neu formatiert und/oder Schemakomponenten in der Datei umgruppiert.
    - Es werden Kommentare oder Dokumentations-Annotationen in der Datei hinzugefügt, geändert, oder weggenommen.

6.3.2 Regeln zur Vergabe/Erhöhung von Versionsnummern

Die folgenden Regeln verfeinern und konkretisieren die Vorgaben in [Gematik1] zur Vergabe von Versionsnummern von Schemadateien.

6.3.2.1 Erhöhung der Revisionsnummer

Festlegung: Die Revisionsnummer MUSS erhöht werden, wenn die Schemadatei geändert wurde, ohne dass die Änderung Auswirkungen auf die Validierung und Verarbeitung des Inhaltes hat.

Dazu gehören die Änderungen die unter 'Umgruppieren und Kommentieren' beschrieben sind, als auch das Aufteilen eines Schemas in mehrere Komponenten-Schemata und/oder das Auflösen von Komponenten-Schemata.

Festlegung: Die Revisionsnummer des Schemas MUSS ebenfalls erhöht werden, wenn sich die Revisionsnummer (und nur diese!) eines referenzierten Schemas ändert.

Festlegung: Die Revisionsnummer SOLL auf '0' zurückgesetzt werden, wenn sich die Nebenversionsnummer und/oder Hauptversionsnummer ändert.

6.3.2.2 Erhöhung der Nebenversionsnummer

Festlegung: Die Nebenversionsnummer MUSS erhöht werden, wenn Erweiterungen an der XSD-Datei vorgenommen werden, die als kompatibel zu XSD- Dateien mit der gleichen Hauptversionsnummer eingestuft werden.

Festlegung: Die Nebenversionsnummer MUSS außerdem erhöht werden, wenn sich die Nebenversionsnummer (und nur diese!) eines referenzierten Schemas erhöht hat.

Festlegung: Die Nebenversionsnummer SOLL auf '0' zurückgesetzt werden, wenn sich die Hauptversionsnummer ändert.

Die zu einer früheren Version kompatiblen Änderungen umfassen vor allem folgende validierungsrelevanten Aspekte:

- Das Ersetzen mehrerer gleicher lokaler Datentypdefinitionen durch eine Referenz auf einen globalen Datentyp mit der gleichen Definition wie die lokalen Datentypen.
- Das Zusammenfassen von Attributen in globalen Attributgruppen und die Verwendung der Gruppe anstatt der lokalen Attributdefinitionen.[20]
- Das Einschränken des Wertebereichs eines Datentyps (die Menge der zulässigen neuen Werte ist eine echte Teilmenge der Menge der Werte der zuvor zulässigen Werte. D.h. es dürfen keine Werte akzeptiert werden, die vorher nicht akzeptiert wurden)
- Die Einschränkung der Kardinalität entweder durch Erhöhen der min-occurence oder Erniedrigung der max-occurence (nicht jedoch umgekehrt). D.h. ein Attribut/Element das

[20] Im Gegensatz zum Umgruppieren von Definitionen werden durch die beiden Änderungen neue Typen/Definitionen zum Schema hinzugefügt, daher muss die Nebenversionsnummer erhöht werden.

optional war, kann erforderlich gemacht werden – jedoch nicht umgekehrt.

- Das Entfernen von optionalen Attributen/Elementen aus dem Schema.

Die praktische Bedeutung der obigen Definitionen liegt darin, dass man die Grenze zwischen der Erhöhung der Nebenversionsnummer und Hauptversionsnummer anhand der Rückwärtskompatibilität der Instanzdokumente ziehen kann. Ein Instanzdokument, das valide zu einem Schema mit einer höheren Nebenversionsnummer ist, ist auch immer noch valide in Bezug auf die Schemata mit früheren Nebenversionsnummern. Der umgekehrte Fall gilt nicht!

6.3.2.3 Erhöhung der Hauptversionsnummer

Festlegung: Die Hauptversionsnummer MUSS in allen anderen Änderungsfällen erhöht werden.

Festlegung: Die Hauptversionsnummer MUSS erhöht werden, wenn sich die Hauptversionsnummer (und nur diese!) eines referenzierten Schemas erhöht hat.

Zu den Änderungen, die zu einer Änderung der Hauptversionsnummer führen, gehören vor allem:

- Das Hinzufügen von neuen Strukturen (z. Bsp. Elemente/Attribute) und Datentypdefinitionen
- Das Ändern existierender Datenstrukturen (z. Bsp. Umgruppierung der Elemente in einer Sequenz).
- Die Erweiterung der Kardinalität von Schemakomponenten (Erniedrigen der min-occurence oder Erhöhung der max-occurence).
- Das Entfernen vormals erforderlicher Strukturen (z. Bsp. Elemente/Attribute)
- Die Erweiterung des Wertebereichs von Datentypen (Die Menge der akzeptierten Werte ist keine echte Teilmenge mehr der vorher akzeptierten Werte).

6.3.3 Geltungsbereich von Versionsnummern

Wie bereits bei den Namensräumen besprochen, gibt es eigene Geltungsbereiche für Echt- und Testdatenbetrieb. D. h. dass die Versionierung für beide Bereiche unabhängig voneinander erfolgt, und dass eventuell eine Schemaversion aus dem Testdatenbetrieb niemals in die Produktion übergeht.

6.4 Schnittstellen Versionierung (logische Version)

6.4.1 Änderungsaspekte

Die folgende Liste gibt einen Überblick über Änderungen die einen Einfluss auf die Versionierung von Schnittstellen haben können:

- Schemaänderung
  - Ein der Schnittstelle zugrundeliegendes XML-Schema hat sich geändert.

- Änderung der Bedeutung von Werten
  - Die Interpretation eines ganzen Wertebereichs kann sich ändern (z. Bsp. indem man die implizite Maßeinheit ändert: DM->EUR, oder indem ein Schlüsselsystem durch ein anderes ausgetauscht wird: Krankenversicherungsnummer -> Sozialversicherungsnummer)

- Änderung von externen Daten
  - Zur Validierung und Verarbeitung von Verfahrensdaten werden oft externe Datenbestände herangezogen (z. Bsp. Schlüsseltabellen), die regelmäßigen oder unregelmäßigen Änderungen unterworfen sein können.

- Veränderung des Verhaltens bei der Validierung/Verarbeitung
  - Die Regeln zur Validierung komplexer Sachverhalte und/oder Verarbeitung ändern sich, z. Bsp. kann beschlossen werden, dass ein Fachverfahren beim Datenabgleich von Stammdaten keine Fehler mehr liefern darf, wenn Schlüsselinformationen nicht vollständig mit Detailinformationen übereinstimmen (Arztnummer -> falsche Adresse des Arztes, aber korrekte Bankverbindung)

Es gilt dabei zu beachten, dass obige Änderungen (außer dem ersten Punkt) keine Änderung der zugrunde liegenden Schemata erfordern, und somit die Schemaversionsnummer alleine nicht ausreicht, um eine Schnittstellenversion eindeutig zu charakterisieren.

6.4.2 Regeln zur Vergabe/Erhöhung von Versionsnummern

Aufgrund der Zielstellung dieses Dokumentes können nur Regeln zur Vergabe von logischen Versionsnummern definiert werden, die durch Änderung von Schemaversionen erforderlich sind. Die Regeln zur Änderung der logischen Versionsnummern aufgrund der Änderung der Bedeutung von Werten, Änderung von externen Daten, oder der Veränderung des Verhaltens bei der Validierung/Verarbeitung werden hier nicht vorgegeben und sind vom Verfahren festzulegen.

6.4.2.1 Erhöhung der Revisionsnummer

Festlegung: Die logische Revisionsnummer MUSS erhöht werden, wenn sich die Revisionsnummer mindestens eines der Schnittstelle zugrunde liegenden XML-Schema-Dateien erhöht (und nur diese!).

6.4.2.2 Erhöhung der Nebenversionsnummer

Festlegung: Die logische Nebenversionsnummer MUSS erhöht werden, wenn sich die Nebenversionsnummer mindestens eines der Schnittstelle zugrunde liegenden XML-Schema-Dateien erhöht (und nicht die Hauptversionsnummer).

6.4.2.3 Erhöhung der Hauptversionsnummer

Festlegung: Die logische Hauptversionsnummer MUSS erhöht werden, wenn sich die Hauptversionsnummer mindestens einer der für ein Verfahren verwendeten XML-Schema-Dateien erhöht.

6.4.3 Validierung von Versionsnummern

Die Beziehung zwischen logischer Schnittstellenversionierung und Schemaversionierung erfordert eine Prüfung durch die Schnittstellenanwendungen. Diese Prüfung muss feststellen, ob eine gegebene logische Versionsnummer kompatibel zu dem Schema des Root-Elementes ist, in dem sie verwendet wird.

Dazu ist es notwendig für jede logische Version eine Abbildung der zu ihr kompatiblen Schema-Versionen zu definieren. Da sich bei jeder signifikanten Änderung des Schemas auch die logische Version ändert, kann man die Abbildung eindeutig definieren

Festlegung: Der Verarbeitungsprozess zu einer Schnittstelle MUSS überprüfen, ob eine gegebene logische Versionsnummer zu der konkret verwendeten XML-Schema-Version kompatibel ist.

6.5 Beispiel

Das folgende Beispiel dient dazu, die Abhängigkeiten zwischen logischer Versionierung und Schemaversionierung zu illustrieren. Die in der rechten Tabellenhälfte angegebenen Versionsnummern resultieren aus der links angegebenen Aktivität. Das Resultat der jeweiligen Aktivität auf die Versionierung ist grafisch hervorgehoben.

Aktivität	Schema-version	Namens-raum-version	Logische Version
1. Erstellung der Schemadatei	**1.0.0**	**1.0**	**1.0.0**
2. Fachliche Änderung hinsichtlich der inhaltlichen Bedeutung von Elementen / Attributen. Die fachspezifische Datenstruktur bleibt davon unberührt. Beispiel: Änderungen / Erweiterungen an externen Schlüsseltabellen (Einschränkung zulässiger Werte)	1.0.0	1.0	**1.0.1**
3. Hinzufügen von Typeinschränkungen Beispiel: Enumerationen (Einschränkung zulässiger Werte)	**1.1.0**	**1.1**	**1.1.0**
4. Hinzufügen neuer Strukturen (basierend auf bestimmten fachlichen Änderungen)	**2.0.0**	**2.0**	**2.0.0**
5. Fachliche Änderung hinsichtlich der inhaltlichen Bedeutung	2.0.0	2.0	**3.0.0**

von Elementen / Attributen. Die fachspezifische Datenstruktur bleibt davon unberührt. Beispiel: Ein Attribut beinhaltet nun andere Informationen: Als Schlüsselattribut enthält ‚key' einen eindeutigen Wert zur Herstellung eines Personenbezugs. Bis einschließlich zur logischen Version 2.0.0 wurde damit die Rentenversicherungsnummer transportiert. Nun wird die KV-Nummer verwendet.			
6. Aus 5. wird der lexikalische Ausdruck in Form einer Regular Expression angepasst.	**2.1.0**	2.1	**3.1.0**
7. Aus 5. werden die weiterführenden Prüfroutinen angepasst. (Außerhalb der Schemadatei)	2.1.0	2.1	**3.1.1**

Daraus ergibt sich die folgende Zuordnungstabelle zwischen logischer Version und Schema-Version:

Logische Version	Schema Version
1.0.0, 1.0.1	1.0.0
1.1.0	1.1.0
2.0.0, 3.0.0	2.0.0
3.1.0, 3.1.1	2.1.0

7 Anleitung zum Erstellen einer empfehlungskonformen XML-Schnittstelle

7.1 Überblick

Die XML-Empfehlung beschreibt einen Rahmen, innerhalb dessen konkrete XML-Schnittstellenimplementierungen im jeweiligen fachlichen Kontext und unter Berücksichtigung des allgemeinen Regelwerks umgesetzt werden können.

Im Folgenden soll beispielhaft der Rahmen für eine Schnittstelle definiert werden. Dazu wird ein fiktives Verfahren mit dem Kennzeichen „EBSP0“ verwendet.

Innerhalb dieses Verfahrens werden 2 Nachrichtentypen „Anfrage“ und „Antwort“ als Nutzdaten modelliert, die zwischen Krankenkassen und Krankenhäusern ausgetauscht werden.

Bei der Modellierung wird auf die Definitionen des GI4X Basis-Schemas, Version 1.0,0 zurückgegriffen.

7.2 Allgemeine Vorgehensweise

7.2.1 Schema/Namensraum Festlegungen

Da in ‚BSP’ 2 verschiedene Nachrichtentypen (=Wurzelelemente) zum Einsatz kommen, definieren wir auch 2 Schemadateien für diese (wg. 5.5.2.3). Wir benennen diese mit den qualifizierenden Namen ‚anfrage’ bzw. ‚antwort’.

Eine erste Analyse der zu übertragenden Daten hat außerdem ergeben, dass in beiden Nachrichtentypen dieselben Stammdaten für die Krankenkasse und das Krankenhaus modelliert werden müssen. Um diese redundanzfrei modellieren zu können, benötigen wir eine Schema-Zwischenschicht, die wir – analog zu GI4X – ‚Basis’ nennen, und die von beiden Nachrichtenschemata importiert wird.

Darüber hinaus hat ein BSP Kontrollgremium entschieden, einige BSP spezifische, aber innerhalb der BSP Landschaft verfahrensübergreifende Datenstrukturen zu definieren, u.a. für binär zu übertragende Dokumente. Für die dafür nötigen Schemadateien wurde die Verfahrenskennung ‚BSP’ verabschiedet.

Da dies die erste Version der Schemata ist, bekommen alle Schemata die Versionsnummer 1.0.0. Hieraus ergeben sich somit die folgenden zu verwendenden Dateinamen und Namensräume:

Dateiname	Namensraum
BSP-basis-1.0.0.xsd	GI4X:/xml-schema/BSP-basis/1.0
EBSP0-basis-1.0.0.xsd	GI4X:/xml-schema/EBSP0-basis/1.0
EBSP0-anfrage-1.0.0.xsd	GI4X:/xml-schema/EBSP0-anfrage/1.0

EBSP0-antwort-1.0.0.xsd	GI4X:/xml-schema/EBSP0-antwort/1.0

7.2.2 Namensraum-Hierarchie

Alle Schemata importieren die vorgegebenen Definitionen aus ‚GI4X-basis-1.0.0.xsd'.

‚EBSP0-basis-1.0.0.xsd' importiert die Definitionen aus ‚BSP-basis-1.0.0.xsd', und ‚EBSP0-anfrage-1.0.0.xsd' und ‚EBSP0-antwort.1.0.0.xsd' importieren die Definitionen aus ‚BSP-basis-1.0.0.xsd' sowie ‚EBSP0-basis-1.0.0.xsd'.

Daraus ergeben sich die folgenden Schema-Datei Templates:

‚EBSP0-basis-1.0.0.xsd':

```
<xs:schema
        targetNamespace="GI4X:/xml-schema/EBSP0-basis/1.0"
        xmlns:xs            ="http://www.w3.org/2001/XMLSchema"
        xmlns:GI4X-basis    ="GI4X:/xml-schema/GI4X-basis/1.0"
        xmlns:BSP-basis     ="GI4X:/xml-schema/BSP-basis/1.0"
        xmlns:EBSP0-basis   ="GI4X:/xml-schema/EBSP0-basis/1.0"
        elementFormDefault  ="qualified" attributeFormDefault ="unqualified"
        version             ="1.0.0">
    <xs:import
        namespace           ="GI4X:/xml-schema/GI4X-basis/1.0"
        schemaLocation      ="GI4X-basis-1.0.0.xsd"
    />
    <xs:import
        namespace           ="GI4X:/xml-schema/BSP-basis/1.0"
        schemaLocation      ="BSP-basis-1.0.0.xsd"
    />
<!—lokale Definitionen ... -->
</xs:schema>
```

‚EBSP0-anfrage-1.0.0.xsd' ‚':

```
<xs:schema
        targetNamespace="GI4X:/xml-schema/EBSP0-anfrage/1.0"
        xmlns:xs            ="http://www.w3.org/2001/XMLSchema"
        xmlns:GI4X-basis    ="GI4X:/xml-schema/GI4X-basis/1.0"
        xmlns:BSP-basis     ="GI4X:/xml-schema/BSP-basis/1.0"
        xmlns:EBSP0-basis   ="GI4X:/xml-schema/EBSP0-basis/1.0"
        xmlns:EBSP0-anfrage ="GI4X:/xml-schema/EBSP0-anfrage/1.0"
        elementFormDefault  ="qualified" attributeFormDefault ="unqualified"
```

```
        version             ="1.0.0">
    <xs:import
        namespace       ="GI4X:/xml-schema/GI4X-basis/1.0"
        schemaLocation      ="GI4X-basis-1.0.0.xsd"
    />
    <xs:import
        namespace       ="GI4X:/xml-schema/BSP-basis/1.0"
        schemaLocation      ="BSP-basis-1.0.0.xsd"
    />
    <xs:import
        namespace       ="GI4X:/xml-schema/EBSP0-basis/1.0"
        schemaLocation      ="EBSP0-basis-1.0.0.xsd"
    />
<!—lokale Definitionen … -->
</xs:schema>
```

Das Template für ‚EBSP0-antwort-1.0.0.xsd' unterscheidet sich von obigem Template nur im Attribut targetNamespace (*GI4X:/xml-schema/EBSP0-antwort/1.0*) und definiert für diesen Wert den Präfix ‚xmlns:EBSP0-antwort' statt ‚xmlns:EBSP0-anfrage'.

7.2.3 Modellierung statischer und dynamischer Wertelisten

Die ‚Anfrage' Nachricht wird mit einer Priorität (statische Werteliste) versehen, deren Werte zwischen 0 und 4 liegen kann, oder den Wert ‚wichtig' enthalten darf. Die Modellierung dieses Datentyps wird über das XML-Schema ‚Union' Konstrukt ermöglicht:

```
    <xs:simpleType name="Prioritaet_Unn">
        <xs:union>
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="wichtig"/>
                </xs:restriction>
            </xs:simpleType>
            <xs:simpleType>
                <xs:restriction base="xs:int">
                    <xs:minInclusive value="0"/>
                    <xs:maxInclusive value="4"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:union>
    </xs:simpleType>
```

In ‚BSP-basis' sind bereits vordefinierte Datentypen für die Krankenkassen/Krankenhaus Institutskennzeichen vorgegeben (dynamische Wertelisten), deren Definitionen im Kommentar je einen Verweis auf eine im WWW verfügbare Liste der aktuell gültigen IK-Nummern enthält:

```
<xs:simpleType name="Krankenkasse_IK_Stp">
    <xs:annotation>
        <xs:documentation source="http://www.gkv.de/Krankenkassenliste.html"
            >Krankenkassen Institutskennzeichen</xs:documentation>
    </xs:annotation>
    <xs:restriction base="GI4X-basis:IK_Stp"/>
</xs:simpleType>
<xs:simpleType name="Krankenhaus_IK_Stp">
    <xs:annotation>
        <xs:documentation source="http://www.gkv.de/Krankenhausliste.html"
            >Krankenhaus Institutskennzeichen</xs:documentation>
    </xs:annotation>
    <xs:restriction base="GI4X-basis:IK_Stp"/>
</xs:simpleType>
```

7.2.4 Modellierung der Stammdaten für Krankenkasse und Krankenhaus

Die komplexen Typen für die Krankenhaus/Krankenkasse Stammdaten sind in ‚EBSP0-basis-1.0.0.xsd' wie folgt definiert:

```
<xs:complexType name="Krankenkasse_Ctp">
    <xs:annotation>
        <xs:documentation>Stammdaten zu Krankenkasse</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="IK" type="BSP-basis:Krankenkasse_IK_Stp"/>
        <xs:element name="Name">
            <xs:simpleType>
                <xs:restriction base="GI4X-basis:ISO88591_Stp">
                    <xs:maxLength value="50"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="Krankenhaus_Ctp">
    <xs:annotation>
        <xs:documentation>Stammdaten zu Krankenhaus</xs:documentation>
    </xs:annotation>
    <xs:sequence>
```

```
        <xs:element name="IK" type="BSP-basis:Krankenhaus_IK_Stp"/>
        <xs:element name="Name">
            <xs:simpleType>
                <xs:restriction base="GI4X-basis:ISO88591_Stp">
                    <xs:maxLength value="40"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:annotation>
    <xs:documentation>Stammdaten zu Krankenkasse</xs:documentation>
</xs:annotation>
<xs:complexType>
    <xs:sequence>
        <xs:element name="IK" type="BSP-basis:Krankenkasse_IK_Stp"/>
        <xs:element name="Name">
            <xs:simpleType>
                <xs:restriction base="GI4X-basis:ISO88591_Stp">
                    <xs:maxLength value="50"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

Zu beachten ist hierbei, wie die Typdefinitionen aus BSP-Basis über Präfix referenziert werden und die beiden Wertebereiche von ‚Name' auf ISO 8859-1 aus dem GI4X Basis-Schema festgelegt und über die ‚maxLength' Fassette eingeschränkt wird!

7.2.5 Modellierung der ‚Anfrage' und ‚Antwort' Nachrichten

Die Arbeitsgruppe des Fachverfahrens hat entschieden, dass in einer Datenlieferung mehrere Anfragen und Antworten (höchstens aber 100) als ‚Paket' übertragen werden können. Um den Wechsel zwischen Verfahrensversionen zu vereinfachen, können die Anfragen/Antworten zu unterschiedlichen logischen Verfahrensversionen gehören. Diese Anforderungen werden in den Schemata wie folgt modelliert:

‚EBSP0-anfrage-1.0.0.xsd':

```
<xs:element name="Datenlieferung">
    <xs:complexType>
        <xs:sequence>
```

```
            <xs:element name="Datenlieferung_ID" type="xs:string"/>
            <xs:element name="Anfrage" type="Anfrage_Ctp" maxOccurs="100"/>
            …
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:complexType name="Anfrage_Ctp">
    <xs:complexContent>
        <xs:extension base="GI4X-basis:Wurzel_Ctp">
            …
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

‚EBSP0-antwort-1.0.0.xsd':

```
<xs:element name="Datenlieferung">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Antwort" type="Antwort_Ctp" maxOccurs="100"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:complexType name="Antwort_Ctp">
    <xs:complexContent>
        <xs:extension base="GI4X-basis:Wurzel_Ctp">
        …
        </xs:extension>
    </xs:complexContent>
    </xs:complexType>
```

Die Ableitung der ‚Anfrage'/'Antwort' Element-Typen aus dem Basis-Typ ‚GI4X-basis:Wurzel_Ctp'[21] durch Erweiterung führt dazu, dass diese das Attribut ‚logische_version' aus dem Basis-Typ erben. Das Element ‚Datenlieferung' dient hier beispielhaft als Transportformat-‚Provisorium' und soll später idealerweise durch ein verfahrensübergreifendes GI4X Transportformat ersetzt werden. Deshalb erlauben wir an dieser Stelle auch die mehrfache Verwendung des eigentlichen Wurzel-Elementes (für die Nachrichten) innerhalb des Transportformat-Containers und verzichten auf die sonst verpflichtende Festlegung, dass ‚Datenlieferung' als Transportformat-Wurzel ein Attribut ‚logische_version' enthalten muss.

[21] Die Festlegung des GI4X-basis:Wurzel_Ctp Typs ist nicht Aufgabe dieser Richtlinie, sondern wird im Rahmen der Data-Dictionary Arbeiten der UAG spezifiziert. Die Verwendung illustriert die Erweiterung von komplexen Typen aus einem anderen Schema. Die konkrete Definition des Basis-Typs kann in GI4X-basis-1.0.0.xsd eingesehen werden.

Zur Identifikation des Empfängers und Senders enthalten beide Elemente die Stammdaten des anfragenden Krankenhauses bzw. der angefragten Krankenkasse, deren Definition bereits in ‚EBSP0-basis.1.0.0.xsd' gegeben ist und über eine Element-Referenz eingebettet werden.

Jede ‚Anfrage' Datenlieferung bekommt eine eindeutige Identität, die vom Sender vergeben wird, und jede Anfrage eine (innerhalb des Paketes) eindeutige Positionsnummer.
Um eine eindeutige Abbildung einer Antwort auf eine Anfrage zu gewährleisten, muss eine Antwort daher immer die Datenlieferung-ID und lokale Position der Anfrage referenzieren.

Der eigentliche Inhalt der Anfrage / Antwort kann entweder als Klartext oder als eingebettetes Binär-Dokument (dessen Definition bereits in ‚BSP-basis-1.0.0.xsd' vorgegeben ist) übermittelt werden.

Das ‚Anfrage' Element, ist dann wie folgt über den komplexen Typ ‚Anfrage_Ctp' definiert:

```
<xs:complexType name="Anfrage_Ctp">
    <xs:complexContent>
        <xs:extension base="GI4X-basis:Wurzel_Ctp">
            <xs:sequence>
                <xs:element name="Anfrage_ID">
                    <xs:simpleType>
                        <xs:restriction base="xs:int">
                         <xs:minInclusive value="0"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
                <xs:element name="Krankenhaus" type="EBSP0-basis:Krankenhaus_Ctp"/>
                <xs:element name="Krankenkasse" type="EBSP0-basis:Krankenkasse_Ctp"/>
                <xs:element name="Prioritaet" type="Prioritaet_Unn"/>
                <xs:choice>
                    <xs:element name="Klartext" type="GI4X-basis:ISO88591_Text_Stp"/>
                    <xs:element name="Dokument" type="BSP-basis:Dokument_Ctp"/>
                </xs:choice>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Nun wird das Element Datenlieferung noch um ein Identitätscontraint erweitert, das dazu dient, die Eindeutigkeit der Anfrage_ID im Kontext einer Datenlieferung zu prüfen:

```
<xs:element name="Datenlieferung">
    <xs:complexType>
```

```
        ...
    </xs:complexType>
    <xs:unique name="Anfrage_ID_Uqe">
        <xs:selector xpath     ="EBSP0-anfrage:Anfrage" />
        <xs:field xpath        ="EBSP0-anfrage:Anfrage_ID" />
    </xs:unique>
</xs:element>
```

Hinweis: Hinweis: Der Präfix in XPath muss hier angegeben werden, da beide Elemente im Schema Namensraum sind!

Die Prüfung der Krankenhausnummer soll anhand einer externen Schlüsseltabelle erfolgen, deren initiale Version in der Datei ‚EBSP0-ik_krankenhaus_keys-1.xml' zur Verfügung gestellt wird, und eine einfache Liste von IK-Nummern der an dem Verfahren teilnehmenden Krankenhäuser enthält.

Das folgende Fragment (in Datenlieferung) definiert diese Prüfung über XML-Schema-Identitätsconstraints und ein ‚Platzhaltermodell' für die in Instanzdokumenten über XInclude einzubettende Liste:

```
<xs:element name="Datenlieferung">
    <xs:complexType>
        <xs:sequence>
        ...
        <xs:element name="Schluesseltabellen">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="IK_Liste_Krankenhaus">
                    <xs:complexType>
                    <xs:sequence>
                        <xs:element name="IK_Nummer" maxOccurs="unbounded"/>
                    </xs:sequence>
                    </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        </xs:sequence>
    </xs:complexType>
...
    <xs:key name="Krankenhaus_Key">
        <xs:selector xpath="
EBSP0-anfrage:Schluesseltabellen/ EBSP0-anfrage:IK_Liste_Krankenhaus/
```

```
EBSP0-anfrage:IK_Nummer" />
        <xs:field xpath="." />
    </xs:key>

    <xs:keyref name="Krankenhaus_Krf" refer="Krankenhaus_Key" >
        <xs:selector xpath="EBSP0-anfrage:Anfrage/EBSP0-basis:Krankenhaus" />
        <xs:field      xpath="EBSP0-basis:IK" />
    </xs:keyref>
...
```

Hinweis: Hinweis: Zu beachten sind die unterschiedlichen XPath Präfixe im zweiten xs:selector Element, da das ‚Anfrage' Element zwar im Namensraum des Schemas ist, die Krankenhaus Stammdaten hingegen aus EBSP0-basis kommen, und damit der Namespace dieses Schemas für ‚Krankenhaus' verwendet werden muss!

Die Prüfung der Krankenkassen-IK Nummer soll weiterhin durch externe Prüfprogramme erfolgen.

Die Modellierung der Antwort-Nachricht verzichtet auf jedwede Prüfung von Identitäts-Constraints durch XML-Schema-Mittel, so dass sich die komplette Antwort wie folgt in XML-Schema darstellen lässt:

```
<xs:element name="Datenlieferung">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Antwort" type="Antwort_Ctp" maxOccurs="100"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:complexType name="Antwort_Ctp">
    <xs:complexContent>
        <xs:extension base="GI4X-basis:Wurzel">
            <xs:sequence>
                <xs:element name="Anfrage">
                    <xs:complexType>
                        <xs:sequence>
                         <xs:element name="Datenlieferung_ID"/>
                         <xs:element name="Anfrage_ID">
                         <xs:simpleType>
                         <xs:restriction base="xs:int"><xs:minInclusive value="0"/></xs:restriction>
                         </xs:simpleType>
                         </xs:element>
                        </xs:sequence>
```

```
                </xs:complexType>
            </xs:element>
            <xs:element ref="EBSP0-basis:Krankenhaus"/>
            <xs:element ref="EBSP0-basis:Krankenkasse"/>
            <xs:choice>
                <xs:element name="Klartext" type="xs:string"/>
                <xs:element ref="BSP-basis:Dokument"/>
            </xs:choice>
        </xs:sequence>
    </xs:extension>
</xs:complexContent>
</xs:complexType>
```

7.3 Schema Diagramm

Die folgenden Diagramme geben noch einmal einen Überblick über die Struktur der ‚Anfrage' und ‚Antwort' Nachrichten für dieses Beispiel:

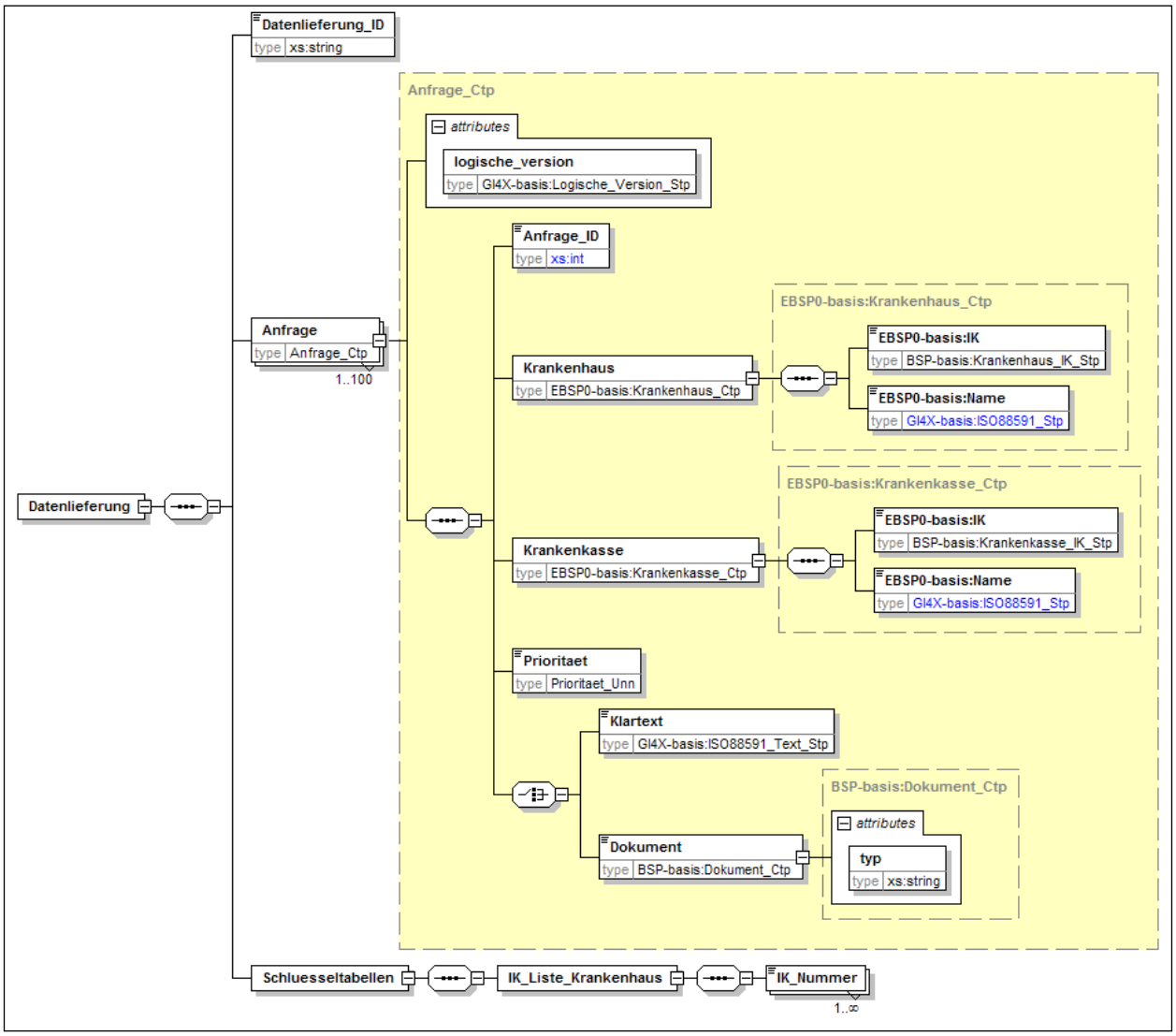


Abbildung 4: Beispiel Schema für ‚Anfrage' Nachricht

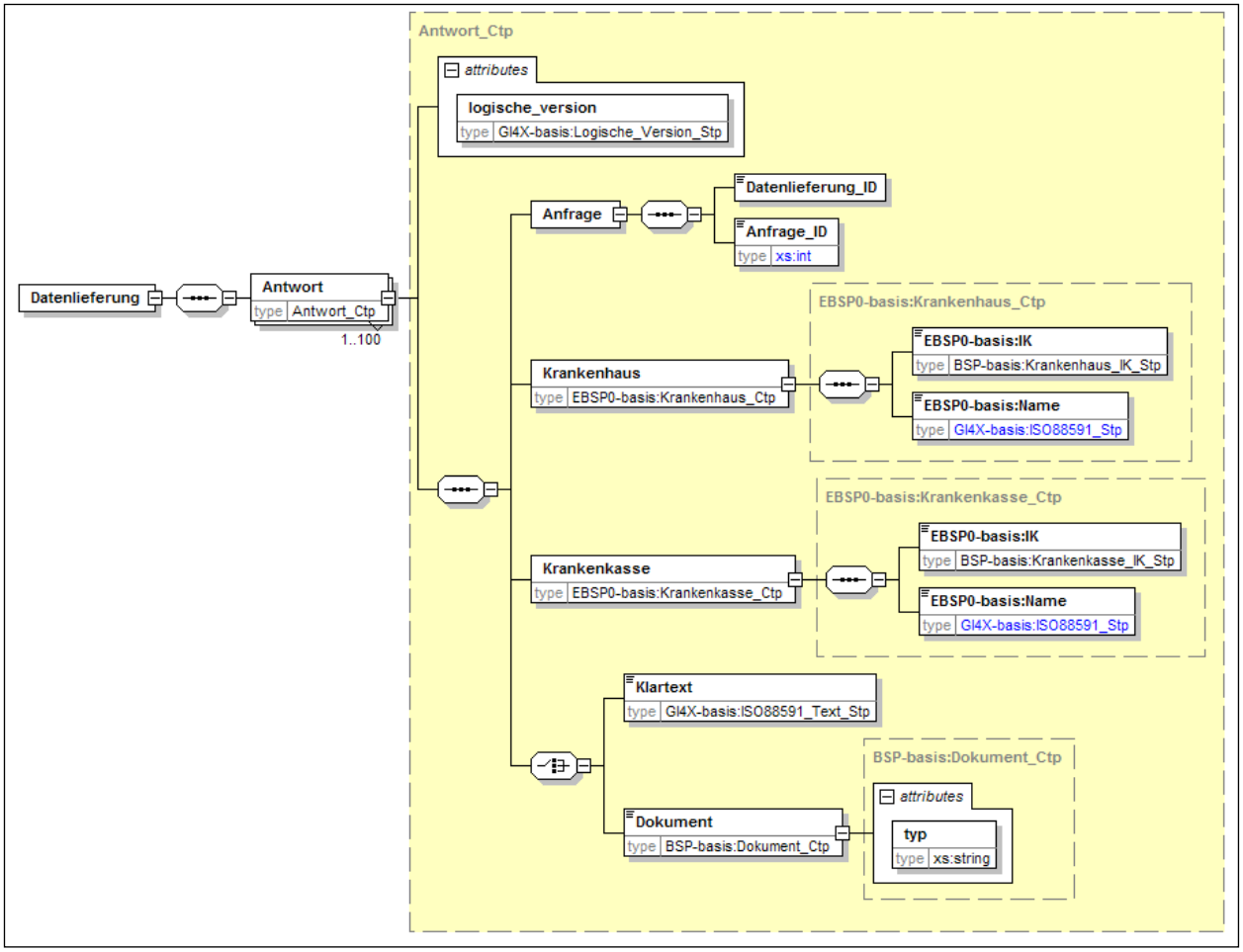


Abbildung 5: Beispiel Schema für ‚Antwort' Nachricht

7.4 Instanz Dokumente

In diesem Abschnitt wird der Vollständigkeit halber je ein Beispiel für ein valides ‚Anfrage' als auch ‚Antwort' Dokument gegeben:

Anfrage:

```
<Datenlieferung
    xmlns              ="GI4X:/xml-schema/EBSP0-anfrage/1.0"
    xmlns:BSP-basis  ="GI4X:/xml-schema/BSP-basis/1.0"
    xmlns:EBSP0-basis     ="GI4X:/xml-schema/EBSP0-basis/1.0"
    xmlns:GI4X-basis ="GI4X:/xml-schema/GI4X-basis/1.0"
    xmlns:xsi          ="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation     ="GI4X:/xml-schema/EBSP0-anfrage/1.0 EBSP0-anfrage-1.0.0.xsd"
>
    <Datenlieferung_ID>KKH-ID1</Datenlieferung_ID>
    <Anfrage logische_version="1.0.0">
        <Anfrage_ID>1</Anfrage_ID>
        <Krankenhaus>
```

```
            <EBSP0-basis:IK>460550131</EBSP0-basis:IK>
            <EBSP0-basis:Name>Kreiskrankenhaus XYZ</EBSP0-basis:Name>
        </Krankenhaus>
        <Krankenkasse>
            <EBSP0-basis:IK>100323099</EBSP0-basis:IK>
            <EBSP0-basis:Name>XYZ BKK</EBSP0-basis:Name>
        </Krankenkasse>
        <Prioritaet>0</Prioritaet>
        <Klartext>Dies ist der Text für die Anfrage</Klartext>
    </Anfrage>
    <Schluesseltabellen>
        <include
            xmlns="http://www.w3.org/2001/XInclude"
            href="EBSP0-ik_krankenhaus_keys-1.xml"/>
    </Schluesseltabellen>
</Datenlieferung>
```

Das obige ‚Anfrage' Beispiel muss XInclude verwenden, um die Schlüsseltabelle der Krankenhaus IK-Nummern zu referenzieren, da die Empfehlung verbietet, diese Daten direkt in Instanzen zu modellieren.

Antwort:

```
<Datenlieferung
    xmlns               ="GI4X:/xml-schema/EBSP0-antwort/1.0"
    xmlns:BSP-basis  ="GI4X:/xml-schema/BSP-basis/1.0"
    xmlns:EBSP0-basis     ="GI4X:/xml-schema/EBSP0-basis/1.0"
    xmlns:GI4X-basis ="GI4X:/xml-schema/GI4X-basis/1.0"
    xmlns:xsi           ="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation     ="GI4X:/xml-schema/EBSP0-antwort/1.0 EBSP0-antwort-1.0.0.xsd">
    <Antwort logische_version="1.0.0">
        <Anfrage>
            <Datenlieferung_ID> KKH-ID1</Datenlieferung_ID>
            <Anfrage_ID>1</Anfrage_ID>
        </Anfrage>
        <Krankenhaus>
            <EBSP0-basis:IK>460550131</EBSP0-basis: IK>
            <EBSP0-basis:Name>Kreiskrankenhaus XYZ</EBSP0-basis:Name>
        </Krankenhaus>
        <Krankenkasse>
            <EBSP0-basis:IK>100323099</EBSP0-basis:IK>
            <EBSP0-basis:Name>XYZ BKK</EBSP0-basis:Name>
        </Krankenkasse>
        <Klartext>Dies ist die antwort auf die obige Anfrage!</Klartext>
```

```
    </Antwort>
</Datenlieferung>
```

7.5 XSD-Dateien für das Beispiel

7.5.1 BSP-basis-1.0.0.xsd

```
<xs:schema
    xmlns:GI4X-basis="GI4X:/xml-schema/GI4X-basis/1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:BSP-basis="GI4X:/xml-schema/BSP-basis/1.0"
    targetNamespace="GI4X:/xml-schema/BSP-basis/1.0"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
    version="1.0.0">
    <xs:import namespace="GI4X:/xml-schema/GI4X-basis/1.0" schemaLocation="GI4X-basis-1.0.0.xsd"/>
    <xs:simpleType name="Krankenkasse_IK_Stp">
        <xs:annotation>
            <xs:documentation source="http://www.gkv.de/Krankenkassenliste.html"
                >Krankenkassen Institutskennzeichen</xs:documentation>
        </xs:annotation>
        <xs:restriction base="GI4X-basis:IK_Stp"/>
    </xs:simpleType>
    <xs:simpleType name="Krankenhaus_IK_Stp">
        <xs:annotation>
            <xs:documentation source="http://www.gkv.de/Krankenhaus_Liste.html"
                >Krankenhaus Institutskennzeichen</xs:documentation>
        </xs:annotation>
        <xs:restriction base="GI4X-basis:IK_Stp"/>
    </xs:simpleType>
    <xs:complexType name="Dokument_Ctp">
        <xs:annotation>
            <xs:documentation>Binär übertragenes Dokument</xs:documentation>
        </xs:annotation>
        <xs:simpleContent>
            <xs:extension base="xs:base64Binary">
                <xs:attribute name="typ" type="xs:string" use="required">
                    <xs:annotation>
                        <xs:documentation>Muss den Mime-Type des Dokumentes enthal-
ten</xs:documentation>
                    </xs:annotation>
                </xs:attribute>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
```

```
</xs:schema>
```

7.5.2 EBSP0-basis-1.0.0.xsd

```
<xs:schema
    xmlns:GI4X-basis="GI4X:/xml-schema/GI4X-basis/1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:BSP-basis="GI4X:/xml-schema/BSP-basis/1.0"
    xmlns:EBSP0-basis="GI4X:/xml-schema/EBSP0-basis/1.0"
    targetNamespace="GI4X:/xml-schema/EBSP0-basis/1.0"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified"
    version="1.0.0"
>
    <xs:import namespace="GI4X:/xml-schema/GI4X-basis/1.0" schemaLocation="GI4X-basis-1.0.0.xsd"/>
    <xs:import namespace="GI4X:/xml-schema/BSP-basis/1.0" schemaLocation="BSP-basis-1.0.0.xsd"/>
    <xs:complexType name="Krankenkasse_Ctp">
        <xs:annotation>
            <xs:documentation>Stammdaten zu Krankenkasse</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="IK" type="BSP-basis:Krankenkasse_IK_Stp"/>
            <xs:element name="Name">
                <xs:simpleType>
                    <xs:restriction base="GI4X-basis:ISO88591_Stp">
                        <xs:maxLength value="50"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Krankenhaus_Ctp">
        <xs:annotation>
            <xs:documentation>Stammdaten zu Krankenhaus</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="IK" type="BSP-basis:Krankenhaus_IK_Stp"/>
            <xs:element name="Name">
                <xs:simpleType>
                    <xs:restriction base="GI4X-basis:ISO88591_Stp">
                        <xs:maxLength value="40"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
        </xs:sequence>
```

```
    </xs:complexType>
</xs:schema>
```

7.5.3 EBSP0-anfrage-1.0.0.xsd

```
<xs:schema
    xmlns:GI4X-basis="GI4X:/xml-schema/GI4X-basis/1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="GI4X:/xml-schema/EBSP0-anfrage/1.0"
    xmlns:BSP-basis="GI4X:/xml-schema/BSP-basis/1.0"
    xmlns:EBSP0-basis="GI4X:/xml-schema/EBSP0-basis/1.0"
    targetNamespace="GI4X:/xml-schema/EBSP0-anfrage/1.0"
    elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0.0">
    <xs:import namespace="GI4X:/xml-schema/GI4X-basis/1.0" schemaLocation="GI4X-basis-1.0.0.xsd"/>
    <xs:import namespace="GI4X:/xml-schema/BSP-basis/1.0" schemaLocation="BSP-basis-1.0.0.xsd"/>
    <xs:import namespace="GI4X:/xml-schema/EBSP0-basis/1.0" schemaLocation="EBSP0-basis-
1.0.0.xsd"/>

    <xs:element name="Datenlieferung">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Datenlieferung_ID" type="xs:string"/>
                <xs:element name="Anfrage" type="Anfrage_Ctp" maxOccurs="100"/>
                <xs:element name="Schluesseltabellen">
                    <xs:complexType>
                        <xs:sequence>
                         <xs:element name="IK_Liste_Krankenhaus">
                         <xs:complexType>
                              <xs:sequence>
                                   <xs:element name="IK_Nummer" maxOccurs="unbounded"/>
                              </xs:sequence>
                         </xs:complexType>
                         </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
        <xs:unique name="Anfrage_ID_Uqe">
            <xs:selector xpath="EBSP0-anfrage:Anfrage"/><xs:field xpath="EBSP0-anfrage:Anfrage_ID"/>
        </xs:unique>
        <xs:key name="Krankenhaus_Key">
            <xs:selector xpath=
    "EBSP0-anfrage:Schluesseltabellen/EBSP0-anfrage:IK_Liste_Krankenhaus/EBSP0-anfrage:IK_Nummer"/>
            <xs:field xpath="."/>
```

```
        </xs:key>
        <xs:keyref name="Krankenhaus_Krf" refer="Krankenhaus_Key">
            <xs:selector xpath="EBSP0-anfrage:Anfrage/EBSP0-anfrage:Krankenhaus"/>
            <xs:field xpath="EBSP0-basis:IK"/>
        </xs:keyref>
    </xs:element>

    <xs:simpleType name="Prioritaet_Unn">
        <xs:union>
            <xs:simpleType>
                <xs:restriction base="xs:string"><xs:enumeration value="wichtig"/></xs:restriction>
            </xs:simpleType>
            <xs:simpleType>
                <xs:restriction base="xs:int">
                    <xs:minInclusive value="0"/><xs:maxInclusive value="4"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:union>
    </xs:simpleType>

    <xs:complexType name="Anfrage_Ctp">
        <xs:complexContent>
            <xs:extension base="GI4X-basis:Wurzel_Ctp">
                <xs:sequence>
                    <xs:element name="Anfrage_ID">
                        <xs:simpleType>
                         <xs:restriction base="xs:int"><xs:minInclusive value="0"/></xs:restriction>
                        </xs:simpleType>
                    </xs:element>
                    <xs:element name="Krankenhaus" type="EBSP0-basis:Krankenhaus_Ctp"/>
                    <xs:element name="Krankenkasse" type="EBSP0-basis:Krankenkasse_Ctp"/>
                    <xs:element name="Prioritaet" type="Prioritaet_Unn"/>
                    <xs:choice>
                        <xs:element name="Klartext" type="GI4X-basis:ISO88591_Text_Stp"/>
                        <xs:element name="Dokument" type="BSP-basis:Dokument_Ctp"/>
                    </xs:choice>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:schema>
```

7.5.4 EBSP0-antwort-1.0.0.xsd

```
<xs:schema
```

```
    xmlns:EBSP0-basis="GI4X:/xml-schema/EBSP0-basis/1.0"
    xmlns:BSP-basis="GI4X:/xml-schema/BSP-basis/1.0"
    xmlns="GI4X:/xml-schema/EBSP0-antwort/1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:GI4X-basis="GI4X:/xml-schema/GI4X-basis/1.0"
    targetNamespace="GI4X:/xml-schema/EBSP0-antwort/1.0"
    elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0.0"
>
    <xs:import namespace="GI4X:/xml-schema/GI4X-basis/1.0" schemaLocation="GI4X-basis-1.0.0.xsd"/>
    <xs:import namespace="GI4X:/xml-schema/BSP-basis/1.0" schemaLocation="BSP-basis-1.0.0.xsd"/>
    <xs:import namespace="GI4X:/xml-schema/EBSP0-basis/1.0" schemaLocation="EBSP0-basis-
1.0.0.xsd"/>

    <xs:element name="Datenlieferung">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Antwort" type="Antwort_Ctp" maxOccurs="100"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="Antwort_Ctp">
        <xs:complexContent>
            <xs:extension base="GI4X-basis:Wurzel_Ctp">
                <xs:sequence>
                    <xs:element name="Anfrage">
                        <xs:complexType>
                         <xs:sequence>
                         <xs:element name="Datenlieferung_ID"/>
                         <xs:element name="Anfrage_ID">
                              <xs:simpleType>
                                   <xs:restriction base="xs:int">
                                        <xs:minInclusive value="0"/>
                                   </xs:restriction>
                              </xs:simpleType>
                         </xs:element>
                         </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="Krankenhaus" type="EBSP0-basis:Krankenhaus_Ctp"/>
                    <xs:element name="Krankenkasse" type="EBSP0-basis:Krankenkasse_Ctp"/>
                    <xs:choice>
                        <xs:element name="Klartext" type="GI4X-basis:ISO88591_Text_Stp"/>
                        <xs:element name="Dokument" type="BSP-basis:Dokument_Ctp"/>
```

```
                    </xs:choice>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:schema>
```

8 Literaturverzeichnis

[Constraints1] http://www.edition-w3c.de/TR/2001/REC-xmlschema-1-20010502/#cIdentity-constraint_Definitions

[Constraints2] http://www.edition-w3c.de/TR/2001/REC-xmlschema-1-20010502/#coss-identity-constraint

[gematik1] Spezifikation von Versionsnummern in Schnittstellenspezifikationen und Software-Komponenten, http://www.gematik.de/upload/gematik_GA_Spezifikation_Versionsnummern_V1_1_0_2303.pdf

[RegExpr] http://www.w3.org/TR/xmlschema-2/#regexs

[RFC 2119] Best Current Practice - Key words for use in RFCs to Indicate Requirement Levels, http://www.ietf.org/rfc/rfc2119.txt

[RFC 2396] Uniform Resource Identifiers (URI): Generic Syntax,http://www.ietf.org/rfc/rfc2396.txt

[Whitespace] http://www.w3.org/TR/xmlschema-2/#rf-whiteSpace

[XML-Encryption] http://www.w3.org/Encryption/2001/

[XML-Schema] http://www.w3.org/XML/Schema

[XML-Signature] http://www.w3.org/Signature/