SWISS MEDICAL DATA EXCHANGE

Technische Dokumentation zur Anwendung von SMEEX

Beschreibung der Schnittstellen zum Framework,
smeexIDManager und smerfIDManager

Version 1.0
Stand Beta

Impressum

Verband Schweizerischer Fachhäuser für Medizinal-Informatik (VSFM)
c/o Vitodata AG, Deisrütistrasse 10
CH-8472 Oberohringen bei Winterthur

Telefon: +41 (0)52 320 55 55
Telefax: +41 (0)52 320 55 66
E-Mail: info@vsfm.ch
Internet: www.vsfm.ch, www.smeex.ch, www.smerf.ch

Vorstand:

Reto Mettler
CEO Vitodata AG
Projektleitung SMEEX

Autoren: Reto Mettler, Viola Ehrensperger

1. Auflage im April 2010

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Code-Beispiele

Glossar

BAKOM	Bundesamt für Kommunikation
CDA	Die **C**linical **D**ocument **A**rchitecture ist ein von *HL7* erarbeiteter auf XML basierender Standard für den Austausch und die Speicherung klinischer Inhalte. Dabei entspricht ein CDA-Dokument einem klinischen Dokument (z.B. Arztbrief, Befundbericht). Es erfolgt keine Zusammenfassung mehrerer Dokumente wie in einer Patientenakte. (Quelle: Wikipedia)
CDA-CH	Erweiterung der *CDA* durch Schweiz Spezifika, baut auf dem VHitG Arztbrief auf.
DICOM	**D**igital **I**maging and **Co**mmunications in **M**edicine ist ein offener Standard zum Austausch von Informationen in der Medizin. Diese Informationen können beispielsweise digitale Bilder, Zusatzinformationen wie Segmentierungen, Oberflächendefinitionen oder Bildregistrierungen sein. (Quelle: Wikipedia)
eCH	eCH fördert, entwickelt und verabschiedet E-Government-Standards. eCH Ziele richten sich nach der E-Government Strategie der Schweiz (Quelle: http://www.ech.ch)
eHealth-CH	1. Unter dem Begriff “eHealth“ werden alle elektronischen Gesundheitsdienste zusammengefasst, siehe http://www.e-health-suisse.ch (Koordinationsstelle Bund-Kantone) und http://www.ig-ehealth.ch (Interessengemeinschaft eHealth) 2. Knoten Name des OID Knotens des Schweizerischen Gesundheitswesens (OID 2.16.756.5.30).
File	Synonym für Datei, Standardbezeichnung für Dateien bzw. Files im vorliegenden Dokument
HL7	**H**ealth **L**evel **7** ist eine Gruppe internationaler Standards für den Austausch von Daten zwischen Organisationen im Gesundheitswesen und deren Computersystemen. HL7 wird als Bezeichnung für die Organisation verwendet, die Standards im Gesundheitswesen entwickelt und unterstützt, sowie für die Versionen 2.x und die Version 3 der Standards und anderer Standards, die von den lokalen HL7-Organisationen in 30 Ländern entwickelt werden. (Quelle: Wikipedia) Die HL7 Benutzergruppe Schweiz verwaltet den Root *OID* Knoten für das Schweizerische Gesundheitswesen (Knotenname: *eHealth-CH, OID* 2.16.756.5.30).

ICPC-2	**I**nternational **C**lassification in **P**rimary **C**are ist ein Klassifizierungssystem für die Hausarztmedizin. ICPC-2 basiert auf dem Episodenkonzept, ermöglicht sowohl Diagnosen/Befunde zu codieren als auch den Grund der Konsultation (RFE=Reason for Encounter) zu erfassen und sogar angeordnete Prozeduren mit zu erfassen. Die aktuelle Version ist Version 2. Die Einführung von ICPC-2 in der Schweiz wird von der Arbeitsgruppe SGAM.Informatics im Auftrag der *SGAM* koordiniert. (Quelle: http://www.icpc.ch und http://www.sgam.ch)
IHE	**I**ntegrating the **H**ealthcare **E**nterprise ist eine Initiative von Anwendern und Herstellern mit dem Ziel den Informationsaustausch zwischen IT-Systemen im Gesundheitswesen zu standardisieren und zu harmonisieren. Die Umsetzung der medizinischen Prozessabläufe zwischen den Systemen und die Schaffung von Interoperabilität stehen hierbei im Vordergrund. IHE formuliert dazu Anforderungen aus der Praxis in so genannten Use Cases, identifiziert relevante Standards und entwickelt technische Leitfäden, so genannte Profile, mit denen ein Hersteller sein Produkt umsetzen und testen kann. (Quelle: Wikipedia)
LOINC	Die **L**ogical **O**bservation **I**dentifiers **N**ames and **C**odes sind eine Zusammenstellung allgemeingültiger Namen und Identifikatoren zur Bezeichnung von Untersuchungs- und Testergebnissen aus Labor und Klinik. Ziel ist die Erleichterung des elektronischen Datenaustauschs bei der Übermittlung medizinischer Untersuchungsergebnisse und Befunddaten. Für den Austausch strukturierter Dokumente (*CDA*) und Nachrichten wird die Verwendung von LOINC empfohlen von *HL7* und *DICOM*. Die Terminologie wird ständig erweitert und regelmässig in Form einer Datenbank publiziert, die Pflege und Dokumentation der LOINC-Datenbank liegt beim Regenstrief Institute (http://www.regenstrief.org). (Quelle: Wikipedia)
Mono	**Mono** ist eine .NET-kompatible Entwicklungs- und Laufzeitumgebung für plattformunabhängige Software, basierend auf dem Common Language Infrastructure-Standard. Das Open-Source-Projekt wird hauptsächlich von Novell betrieben. Unterstützte Betriebssysteme: Unix/BSD-Derivate (darunter Mac OS X), GNU/Linux, Windows und Solaris 8 (http://www.mono-project.com) (Quelle: Wikipedia)

OHF	Eclipse **O**pen **H**ealthcare **F**ramework, ein Projekt innerhalb von Eclipse, das die Technology der Medizinal-Informatik vorantreiben soll. Das Projekt besteht aus erweiterbaren Frameworks und Werkzeugen, die die Nutzung bestehender und aufkommender Standards unterstützen, um kompatible open source Infrastrukturen zu stärken und damit Hürden bei der Integration zu verkleinern. Aktuell stellt unter anderem OHF Werkzeuge und Frameworks für *HL7* und *IHE* zur Verfügung. (Quelle: http://www.eclipse.org/ohf/)
OID	Weltweit eindeutige Objekt ID für Informationsobjekte. ISO/IEC 9834/1 normiert. OIDs unter dem Schweizer Länderknoten werden vom *BAKOM* zugewiesen.
SGAM	**S**chweizerische **G**esellschaft für **A**llgemein**M**edizin (http://www.sgam.ch/)
SMEEX	**S**wiss **Me**dical data **Ex**change Projekt des *VSFM* zur Entwicklung eines systemübergreifenden Standards für den Austausch medizinischer Daten. SMEEX ist als Marke im Schweizer Markenregister eingetragen. (Markenregister: http://www.swissreg.ch, Informationen zu Marken: http://www.ige.ch)
smeexID	Objekt ID, die jedes Datenfeld innerhalb des *SMEEX* Datenformats eindeutig identifiziert, entspricht einer erweiterten *OID*-Definition, wobei der smeexID neben der „OID“ Nummer weitere Attribute wie ein Datentyp, ein optionales Pattern und eine Kardinalität zugewiesen werden. Die Root *OID* der smeexIDs ist 2.16.756.5.30.1.106.1.10.
SMERF	**S**wiss **ME**dical **R**e**F**erencesystem, definiert einen Nomenklaturkatalog für die Art eines klinischen Resultats auf Basis einer OID-ähnlichen Struktur. Der SMERF Katalog ist in Zusammenhang mit dem SMEEX Projekt entwickelt worden mit dem Ziel, klinische Resultate so zu codieren, dass die Art des klinischen Resultates möglichst vollständig erfasst ist.
smerfID	Objekt ID, die jedes Element innerhalb des *SMERF* eindeutig identifiziert. Sie entspricht einer erweiterten *OID*-Definition, wobei der smerfID neben der „OID“ Nummer weitere Attribute wie die UCUM Einheit und äquivalente *LOINC*-Codes zugewiesen werden. Die Root *OID* der smerfIDs ist 2.16.756.5.30.1.106.1.13.

SNOMED	**S**ystematisierte **No**menklatur der **Med**izin (engl.: **S**ystematized **No**menclature of Human and Veterinary **Med**icine) ist die bedeutendste Nomenklatur der Medizin. Ziel ist es, medizinische Aussagen so zu indizieren, dass die inhaltlichen Elemente der Aussage vollständig erfasst sind, wodurch sehr spezielle Suchanfragen mit hohem Recall (Vollzähligkeit) und hoher Präzision (Relevanz) beantwortet werden können. Des Weiteren existieren Querverbindungen zu Wissenssammlungen und Literaturdatenbanken. SNOMED gibt es aktuell in den Versionen SNOMED II, SNOMED 3 und SNOMED CT. (Quelle: Wikipedia)
UCUM	Der **U**nified **C**ode for **U**nits of **M**easure ist ein regelbasiertes Kodierungssystem für Masseinheiten. Zweck dieses Standards ist der elektronische Datenaustausch von physikalischen Grössen. Die *OID* für das Kodierungssystem UCUM ist 2.16.840.1.113883.6.8. Die formale Definition, Codetabellen und Beispielimplementierungen finden sich auf den UCUM-Seiten des Regenstrief Institute (http://www.regenstrief.org) (Quelle: Wikipedia)
UMLS	**U**nified **M**edical **L**anguage **S**ystem ist ein Projekt der US National Library of Medicine, das Terminologien biomedizinischer Ressourcen wie Online-Datenbanken und medizinischen Wörterbüchern aneinander angleichen will. Dies geschieht, indem Konzepte und Relationen verschiedener vorhandener Datenbanken miteinander in Beziehung gebracht werden. (Quelle: Wikipedia)
VHitG	**V**erband der **H**ersteller von **IT**-Lösungen für das **G**esundheitswesen e.V. Im VHitG sind die wichtigsten IT-Hersteller Deutschlands aus dem Bereich der Krankenhaussoftware, Arztpraxissoftware und anderer Softwareprodukte vertreten. (Quelle: http://www.vhitg.de)
VHitG Arztbrief	Vom VHitG entwickelter Standard für die elektronische Übermittlung von Arztbriefen. Der Arztbrief an sich soll Informationen über Erkrankung, Therapie und Empfehlungen bei einer Überweisung von einem Arzt zum anderen enthalten, bisher meist in Papierform gehalten.
VSFM	**V**erband **S**chweizerischer **F**achhäuser für **M**edizinal-Informatik; Auftraggeberin für das Projekt *SMEEX.* Dem VSFM ist die *OID* 2.16.756.5.30.1.106 zugewiesen.
XDS	**Cross** enterprise **D**ocument **S**haring (XDS) ist ein von der *IHE* definiertes Profil, das den Austausch klinischer Dokumente zwischen Institutionen vereinfachen soll. (Quelle: Wikipedia)

Dokumenten Management

Versions-Kontrolle

Version	Datum	Autor	Beschreibung
1.0 beta	16.04.2010	VEH	Initiale Version

Dokumenten Verteiler

Verteiler-Liste

Version	Datum	Empfänger	Format (Hard / Soft)
1.0 beta	16.04.2010	www.smeex.ch	Download Quelle, Soft

1 Einleitung

1.1 Vorwort an den Leser

Das Thema Datenaustausch im Gesundheitswesen beschäftigt die Branche schon seit geraumer Zeit. Inhaltlich wird viel darüber diskutiert und spekuliert. Dabei geht es darum, wie die technischen Anforderungen zu definieren sind, damit ein systemübergreifender Datenaustausch zwischen den Leistungserbringern und anderen Akteuren im Umfeld des Gesundheitswesens möglich wird. Die SMEEX Initiative ist im Jahre 2008 durch den Verband Schweizerischer Fachhäuser für Medizinal-Informatik (VSFM) lanciert worden. Hauptziel der Initiative ist es; den Standardisierungsprozess für den Datenaustausch im Schweizerischen Gesundheitswesen voranzutreiben. Gleichzeitig sollen die notwendigen technischen Rahmenbedingungen geschaffen werden, damit Systemintegrationen die „Datenaustauschfähigkeit“ auf einfache Art und Weise bestehende Systeme eingebaut werden kann. Konkrete Softwarekomponenten sollen die geplante Funktionalität im gesamten Umfang demonstrieren.
Heute im Jahre 2010 stehen die als Zielsetzung formulierten Softwarekomponenten zur Verfügung. Es sind dies; das SMEEX-Framework für die direkte Integration in eine Branchenapplikation, der smeexIDManager welcher die eineindeutigen Referenzieren der Objekte beheimatet so wie einzelne PlugIns für die Datentransformation und Kommunikation. Die vorliegenden Komponenten sind mit der grösstmöglichen Sorgfalt und Weitsicht entwickelt worden und bilden eine solide Basis. Dennoch ist sich das Entwicklungsteam einig, es handelt sich bei den per April 2010 freigegeben Komponenten um die ersten Versionen (Betas), welche sicherlich noch angepasst, überarbeitet und weiterentwickelt werden müssen.
Die vorliegende Dokumentation beschreibt die einzelnen Komponenten aus einer technischen Sicht und bildet den Integrationsleitfaden für Softwareentwickler und Systemintegratoren, welche sich mit dem Thema SMEEX auseinandersetzen möchten. Es bildet zudem die Einstiegsdokumentation, welche ihrerseits auf weitere Dokumentationen wie Schnittstellen- und Bibliothekbeschreibungen referenziert.
Ich wünsche Ihnen abschliessend viel Spass beim SMEEX-Studium und eine erfolgreiche Implementierung.

Ihr
Reto Mettler

1.2 Schriftkonventionen

Um eine bessere Lesbarkeit zu erreichen, wurden bestimmte Textelemente durch eine spezielle Formatierung hervorgehoben.
Im Folgenden wird jede dieser Hervorhebungen durch ein Beispiel illustriert.

Code

```
Dim a As New Archive("C:\test.smeex")
```

Codebeispiele werden immer in VB.NET angegeben

XML

```
<tag1>
    <tag2>Wert</tag2>
    <tag3/>
</tag1>
```

Kommandozeilenaufruf

```
RegAsm.exe /tlb /codebase Vsfm.SMEEX.DemoDB.Integration.dll
```

Schnittstellen / Klassen im laufenden Text

IStorage

Namespaces

Vsfm.Smeex.Framework.Core

Hinweise/zur besonderen Beachtung

Hinweis:
Dies ist ein Hinweis.

Achtung:
Diesen Hinweis bitte besonders beachten.

Hyperlinks

http://www.vsfm.ch

1.3 Danke

Viele innovative Köpfe haben zum Gelingen des Projekts beigetragen. Auch wenn wir jetzt erst am Anfang der Verbreitung von SMEEX stehen, ist es Zeit um Danke zu sagen. Es ist aufgrund der zahlreichen Personen, die in irgendeiner Form zum Projekterfolg beigetragen haben, nicht möglich alle namentlich aufzuführen. Vorab Ihnen allen ein herzliches Dankeschön für die guten Diskussionen, die kritischen Betrachtungsweisen und letzten Endes für die Konsensfindung im Sinne der gemeinsamen Datenaustauschsache.
Bei den folgenden Projekthauptdarstellern bedanke ich mich im Besonderen. Ihr seid mitverantwortlich, dass die „SMEEX-Brücke“ das tiefe Tal überwinden konnte.

Peter Amherd
Ist massgeblich „schuld“, dass das SMEEX Projekt gestartet wurde. Triebfeder war damals in einer Tiefgarage der grosse Frust über das Nichtweiterkommen im Thema Datenaustausch. Hat unzählige Referate und Diskussionen geführt, um dem Thema Datenaustausch im Markt ein deutliches und markantes Gesicht zu verleihen.

Manuel Dubs
Hat sehr schnell die Notwendigkeit erkannt, das Projekt über den VSFM lanciert und viele wichtige Türen geöffnet.

Freddy Kühne
Ist der technisch geniale Kopf, welcher alle visionären Ansätze auch in die Tat umsetzt. Engineering in Reinkultur!

Stefan Gerber
Rolf Eggenberger
Zwei herausragende Softwarewissenschaftler mit Drive und Hartnäckigkeit. Es brauchte keine grossen Überredungskünste, um diesen beiden Herren der Firma TMR AG ins SMEEX-Boot zu holen. Der smeexIDManager stammt mehrheitlich aus ihrer Feder – eine feine Engineeringleistung! Die Zusammenarbeit ist für das gesamte Projektteam eine echte Bereicherung.

MediData AG
mit dem SMEEX-Entwicklungsteam:
Daniel Bätschmann, Guido Riedweg und Anita de Jong
War die erste Unternehmung, die sich für den SMEEX-Ansatz interessierte und im SMEEX-Boot Platz nahm - schnell und unkompliziert. Half tatkräftig mit das SMEEX-Framework reifen zu lassen und lieferte den Proof-of-Concept im Bereich der PlugIns für die Datentransformation und Kommunikation. Tolle Sache und gute Arbeit! Das Projektteam freut sich auf die weitere Zusammenarbeit.

Viola Ehrensperger
Die sehr gute „Dokumentationsfee“ und meine Co-Autorin, welche die Kohle zu guter Letzt aus dem Feuer holt und das SMEEX-Wissen literarisch zugänglich macht.

2 Abstrakt

Dieses Dokument erläutert den SMEEX Standard, das Framework sowie die Referenzsysteme smeexIDManager und den SMERF (swiss medical reference system). Es vermittelt das nötige Hintergrundwissen, um die Komponenten erfolgreich in Applikationen zu integrieren.
Diese Anleitung richtet sich in erster Linie an Softwareentwickler und an technisch interessierte IT-Fachkräfte. Für IT-Projektleiter bildet dieses Dokument einen Themeneinstieg.

Auch wird eine Demo Applikation vorgestellt, die die Nutzung des SMEEX Frameworks veranschaulicht. Weiter werden zwei Werkzeuge für die Entwicklungs-Unterstützung beschrieben. Dabei handelt es sich zum einen um den smeexIDManager. Der smeexIDManager verwaltet die eindeutigen Referenzen auf Feldebene. Dabei werden die Referenzen von einer Root OID „abgeleitet“ und bilden somit eine eindeutige Identifikation. Nebst den IDs enthält diese Tool die entsprechend notwendige Dokumentation in Prosa-Form.
Beim zweiten Werkzeug handelt es sich um eine UCUM Bibliothek, die Methoden für die Validierung und Konvertierung von Masseinheiten zur Verfügung stellt. Die Funktionalität der UCUM Bibliothek wird mittels einer eigenständigen Demo-Applikation erläutert.

Das Thema Datenaustausch im Schweizerischen Gesundheitswesen ist teils sehr komplex und umfangreich. Mittels des SMEEX Formats lassen sich per se alle Datenstrukturen abbilden. Technisch gesehen können somit alle Daten von einem System in ein anderes – über den SMEEX Datencontainer – überführt werden.
Die Klassifizierung und Codierung – damit Daten auf der Empfangsseite automatisiert eingelesen werden können – bilden die Hauptschwierigkeit. Bestehende Systeme für die medizinische Klassifizierung wie LOINC oder SNOMED sind derweil wenig stark verbreitet, daher musste ein einfaches und erweiterbares „Klassifikationssystem“ eingeführt werden. SMERF übernimmt diese Aufgabe. SMERF steht für swiss medical reference system. Das System bildet eine einfache Abstraktionsebene für die Klassifizierung von medizinischen Daten, welche sich beliebig erweitern lässt.
Bei der Entwicklung wurde darauf geachtet, dass sich die granularste Einheit auf einen/mehrere LOINC-Codes mappen lässt.
Weitere Ausführungen zum Thema Klassifizierung von medizinischen Daten sind dem Kapitel 4.7 zu entnehmen.
Es ist zu beachten, dass sich das SMEEX-System mit all seinen Umsystemen fortlaufend in Entwicklung befindet.
Für weitere Hinweise und Informationen zum Thema SMEEX lesen Sie bitte zudem die aktuellen Infos auf der Webseite.

3 Grundlagen

3.1 Systemvoraussetzungen

SMEEX Framework:

- Mindestens .Net Framework 2.0

PlugIns können allenfalls höhere Versionen des .NET Frameworks benötigen.

Betrieb der DemoDB:

- MS Access 2007

smeexIDManager und smerfIDManager basieren auf einer MS Access 2007 Datenbank, für den Betrieb ist keine Installation von MS Access nötig.

Bezüglich Hardware sind derweil keine restriktiven Systemvoraussetzungen gegeben. Es wird ein handelsübliches Client- oder Serversystem empfohlen. Wichtig dabei ist, dass je grösser die Datenmenge ist, die exportiert oder importiert werden soll, desto mehr RAM benötigt wird. Es empfiehlt sich im Minimum 8 GB RAM auf einem 64 bit System einzusetzen, wobei das System auch auf einer 32 bit Architektur mit 4 GB RAM betrieben werden kann. Die HD-Kapazität ist analog dem Clientsystem zu definieren. Je nach Konfiguration empfiehlt sich, Faktor 2 zu wählen.

3.2 Abgrenzung zu bestehenden Standards

Bei SMEEX handelt es sich um die Definition eines Datenformats für den Austausch von Daten im Medizinalbereich. Der SMEEX Standard beschreibt lediglich Syntax und Semantik des Inhalts eines Datencontainers. Damit ist SMEEX unabhängig von zugrunde liegenden Prozessen oder dem technischen Vorgang, - der Art und Weise - des Austauschs. Bestehende Standards werden nicht konkurrenziert, sondern – wo sinnvoll - integriert.

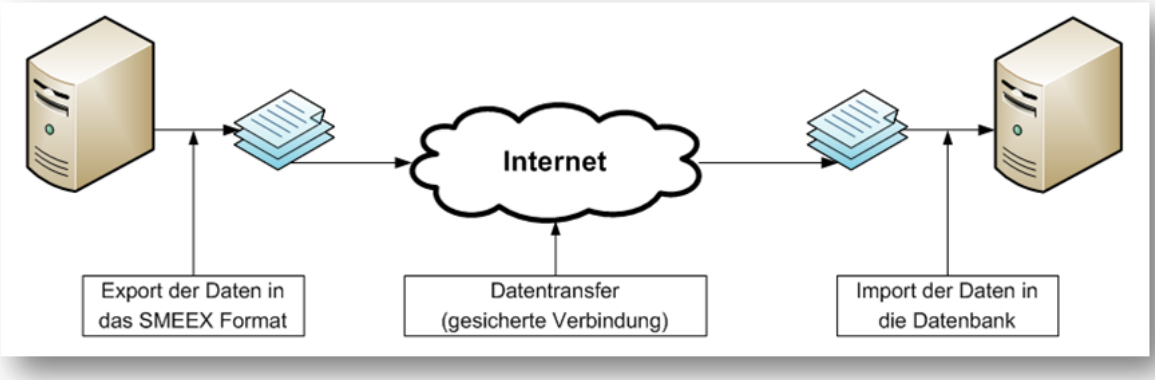

Abbildung 1: Gesamt-Topologie

SMEEX unterscheidet sich von anderen Datenaustauschprojekten im Umstand, dass sowohl patientenbezogene als auch nicht patientenbezogene Daten ausgetauscht werden können. Die heute etablierten Standards konzentrieren sich auf den Austausch von patientenbezogenen Daten in Form von elektronischen Dokumenten. Damit ist es nicht möglich, der Anforderung nach kompletten Systemmigrationen nachzukommen. Hierfür ist es notwendig, „Datenkonstrukte“ für Stammdaten, welche keinen direkten Patientenbezug aufweisen, bereitzustellen.
SMEEX ist also eine Ergänzung und Erweiterung bestehender Standards. (Links zu diesen Standards sind im Anhang unter 7.2.1 zu finden)

3.3 Für .Net Entwickler

Das SMEEX Framework wurde unter .NET entwickelt. Daher kann es direkt aus .NET Projekten referenziert werden.

Die Entwicklungsumgebung bildet MS Visual Studio 2008 mit Zielframework .NET 2.0. Die einzelnen Komponenten werden in Continuous-Builds in Unit-Tests getestet.

3.4 Für C++ Entwickler

Das SMEEX Framework ist in grossen Teilen COM Interop fähig. Eine Registrierung wird nicht automatisch gemacht und muss daher vom Entwickler vor der Nutzung mit Hilfe des Tools *RegAsm.exe* durchgeführt werden.

Beispiel für die Registrierung:

```
RegAsm.exe /tlb /codebase Vsfm.SMEEX.DemoDB.Integration.dll
```

Hierbei muss der Speicherort des Tools *RegAsm.exe* im aktuellen Pfad angegeben sein oder im obigen Aufruf voll qualifiziert werden. Das Tool ist im Verzeichnis des .NET Frameworks 2.0 zu finden (z.B. unter C:\Windows\Microsoft.NET\Framework\v2.0.50727). Da die Versionen 3.0 und 3.5 die CLR von Version 2.0 verwenden, ist in den jeweiligen Verzeichnissen auch das Tool nicht enthalten.

Hinweis:
In der aktuellen Beta-Version des Frameworks sind nur die beiden Bibliotheken Vsfm.SMEEX.DemoDB.Integration.dll und Vsfm.Smeex.Framework.Common.Classes.dll COM-registrierbar. In Version 1.0 des SMEEX Frameworks werden auch die anderen Bibliotheken COM Interop fähig sein.

Über COM Interop kann die Referenz-Implementierung des Frameworks genutzt werden. Anpassungen, wie die Entwicklung eigener PlugIns oder das Ersetzen der Standard UIs (siehe 4.3.6), sind nur eingeschränkt möglich. Diese Erweiterungen basieren auf der Implementierung von Interfaces, das ist nicht in allen Programmiersprachen möglich.

3.5 Für Java Entwickler

Java Entwickler können die SMEEX Bibliothek zum Beispiel über JNI und COM Interop oder über das Tool *JNBridge* ansprechen.

Microsoft stellt auf den MSDN Seiten ein Bespiel-Projekt für den Aufruf von .NET Objekten mit Hilfe des Tools *JNBridge Pro* aus einer Java Anwendung zur Verfügung. Das Beispiel kann unter folgendem Link heruntergeladen werden:
http://www.microsoft.com/germany/msdn/solve/codeclips/library.aspx?id=msdn_de_36042

Details zu den Einsatzmöglichkeiten unter COM Interop siehe 3.4.

3.6 Für Mac Entwickler

Auf einem Mac-System kann das Framework unter Mono genutzt werden. Die Version 2.0 entspricht in weiten Teilen .NET 2.0. Da in Mono 2.0 keine 100%ige Abdeckung der Funktionalität von .NET 2.0 erreicht wurde, ist es möglich, dass für den Betrieb des SMEEX Frameworks nötige Objekte fehlen. Die neueren Technologien von .NET 3.0 und 3.5 wurden noch nicht in Mono implementiert. Falls also PlugIns allenfalls höhere Versionen des .NET Frameworks zwingend benötigen, ist nicht damit zu rechnen, dass sie unter Mono lauffähig sind (Quelle: Wikipedia).
Die Nutzung des SMEEX Frameworks unter Mono ist nicht getestet und wird daher nicht offiziell propagiert.

3.7 Für Linux Entwickler

Unter Linux kann das Framework ebenfalls unter Mono genutzt werden, siehe Kapitel für Mac-Entwickler (3.6).

4 Hauptteil

4.1 Gesamtsicht Architektur

Daten unter verschiedenen Systemen auszutauschen, ja gar zu synchronisieren, gehört zu den komplexeren Themenbereichen der Informatik. Durch die zunehmende Vernetzung im Schweizerischen Gesundheitswesen besteht zunehmend der Bedarf, Daten zwischen den einzelnen Gesundheitsorganisationen auszutauschen. Dieses Bedürfnis ist grundsätzlich nicht neu, denn bereits in anfangs 2000 kam dieses Thema auf. Die Softwarehersteller nahmen sich dem Anliegen des Marktes an und lösten die jeweiligen Problemstellungen projektweise. Durch dieses Vorgehen entstanden im Laufe der Zeit unzählige Datenaustauschschnittstellen und Datenkonverter, welche in der Lage sind Daten aufzunehmen und diese für ein entsprechendes Dritt-System aufzubereiten.

Das Problem liegt im uneinheitlichen Datenformat, sowohl strukturell wie inhaltlich. Konkret bedeutet dies, dass im Schweizerischen Gesundheitswesen kein breit akzeptierter und angewandter Datenaustauschstandard zur Verfügung steht. Durch diese Inexistenz werden die Kosten auf Seiten der Softwarehersteller substantiell getrieben, was sich letzten Endes auf die Produktpreise für den Endverbraucher auswirkt.

Mit dem SMEEX-Ansatz wird das Ziel verfolgt, ein einheitliches Datenaustauschformat für das Schweizerische Gesundheitswesen bereitzustellen. D.h. Datenstruktur wie Dateninhalt sollen definiert werden, wobei durch generische Ansätze die Erweiterbarkeit der Datenstruktur gewährleistet werden soll. Mittels eines Software-Frameworks soll die applikatorische Integration des SMEEX-Standards vereinfacht werden. Es geht bei diesem Ansatz also nicht darum bereits bestehende Datenformate zu verdrängen, sondern diese umfassend zu integrieren.

Der SMEEX-Ansatz repräsentiert die Summe aller im schweizerischen Gesundheitswesen vorhandenen Daten.

Abbildung 2: Gesamtsicht SMEEX Architektur

SMEEX-Framework

Das Framework bildet die eigentliche Software-Komponente mit eindeutigen Schnittstellen zwischen der Branchen-Applikation und dem SMEEX-Standard. Das SMEEX-Framework vereint folgende Funktionen:

- Datenexport / Import gegenüber der Branchen-Applikation, wobei das Framework nur Daten entgegen nimmt resp. bereit stellt. Das Framework verfügt dabei über keinen direkten Datenbankzugriff.
- Die von der Branchen-Applikation erhaltenen Daten werden in das SMEEX-Format transformiert und für die weiteren Aktionen zur Verfügung gestellt.
- PlugIn-Architektur für Datentransformation: mittels dieses PlugIn sollen SMEEX-Daten in gebräuchliche Formate transformiert werden können.
- PlugIn-Architektur für die Datenkommunikation: Dieses PlugIn ermöglicht den webbasierten Datenaustausch zwischen den Akteuren im Schweizerischen Gesundheitswesen.

Die PlugIn-Architektur ist so ausgelegt, dass verschiedene PlugIn *(n an der Zahl)* von unterschiedlichen Herstellern betrieben werden können. Über die detaillierte Funktionalität entscheidet letzten Endes der Hersteller selbst. Einzige Bedingung bleibt, sich an die Framework-Integrationsschnittstellen zu halten (siehe 4.3.4).

SMEEX-Format
Das SMEEX-Format bildet das eigentliche „Herzstück“ des Frameworks. In diesem Format werden die Daten strukturell wie inhaltlich definiert. Technisch gesehen ist das SMEEX-Framework ein komprimiertes Verzeichnis *(ZIP-Archiv)*. In diesem Verzeichnis befinden sich das Dateninhaltsverzeichnis, die eigentlichen Daten als XML-Files und die allenfalls binären Files.

SMEEX-Kommunikator-PlugIn
Dieses PlugIn stellt die elektronische Datenkommunikation sicher und stellt sämtliche Funktionalitäten die für den gesicherten datenaustausch notwendig sind zur Verfügung. Verschiedene Anbieter können einen Kommunikationsdienst entwickeln und im Markt anbieten. Für den Verkehr zwischen SMEEX-Applikationen ist grundsätzlich kein Kommunikator erforderlich (ausser beim Bedarf, einen verschlüsselten, sicheren Kommunikationsweg einzuhalten).

SMEEX-Transformator-PlugIn
Mittels dieses PlugIns können SMEEX-Daten in gebräuchliche Datenformate transformiert werden. Unter gebräuchlich werden derzeit die Formate **c**linical **d**ocument **a**rchitecture (CDA), **cross** enterprise **d**ocument **s**haring (XDS) und **p**ortable **d**ocument **f**ormat (PDF) verstanden. Es ist davon auszugehen, dass im Laufe der Zeit weitere verschiedene Formate implementiert und unterstützt werden.

4.2 Das SMEEX Datenformat

Der SMEEX-Standard - nachfolgend Standard genannt - definiert den medizinischen Datenaustausch im Schweizerischen Gesundheitswesen auf Basis von XML-Strukturen. Diese Strukturen - nachfolgend Archiv genannt - beinhalten:

1. Metadatenverzeichnis für den direkten Datenzugriff, auch als Index bezeichnet
2. Unterordner „data“ mit XML Datenfiles, welche die eigentlichen Daten enthalten (der Standard definiert dabei die Struktur der Datenfiles)
3. Unterordner „annex“ mit binären Datenfiles

Das Archiv-File erhält die Erweiterung „*.smeex“ und ist ein ZIP-Archiv.
Die Metadaten und mindestens ein XML-Datenfile sind zwingend, ansonsten entspricht es nicht dem Standard. Zusätzlich kann eine beliebige Anzahl weiterer XML- und binärer Datenfiles geliefert werden.

Das Metadatenverzeichnis und die XML-Datenfiles sind UTF8 kodiert.

Aufgrund der verwendeten Technologien (GZIP / RFC 1952) ist die Input-Filegrösse auf 4GB beschränkt!

Abbildung 3: SMEEX Datenformat

Metadatenverzeichnis (Index)
Innerhalb des Indexes werden alle technisch notwendigen Informationen für die Verwaltung des Archivs abgelegt. Zudem beinhaltet der Index zusätzliche Informationen, die für die Verarbeitung des Archivs nützlich sein können. Das Metadatenarchiv muss zwingend vorhanden sein.

XML-Datenfiles
Ein strukturierter Datenaustausch erfordert, dass die Daten auch **strukturiert** abgelegt und transportiert werden können. SMEEX verwendet dafür XML-Files, die sowohl die notwendige Flexibilität, als auch den komfortablen Zugriff auf die Daten mit sich bringen.
Es muss mindestens ein Datenfile vorhanden sein. Innerhalb eines SMEEX-Files können mehrere XML-Files vorhanden sein, wodurch nochmals eine zusätzliche logische Strukturierung erreicht werden kann.

Binäre Datenfiles
Nicht alle Daten sind strukturiert verfügbar. So sind die Informationen z.B. in einem Bild in der Regel nicht vernünftig als XML abbildbar. Daher besteht die Möglichkeit, innerhalb von SMEEX-Files binäre Anhänge mitzuschicken, die auf Empfänger-Seite wieder als solche weiter verarbeitbar sind. Binäre Datenfiles ins Archiv einzubinden ist optional.
Neben Bildern können auch PDF-, DOC- oder Video-Formate eingebunden werden. Es ist jedoch zu berücksichtigen, dass solche Informations-Inhalte sehr viel Speicherplatz benötigen können, und deshalb die Verwendung anforderungsspezifisch eingesetzt werden sollte.

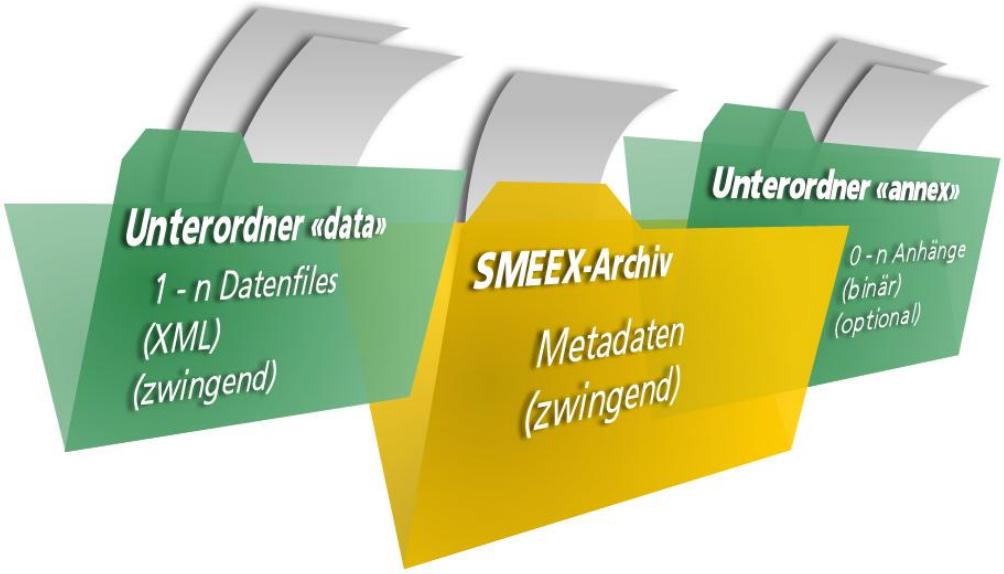

Abbildung 4: File-Struktur des SMEEX Archivs

4.2.1 Filesplitting

Der Mechanismus des Filesplits dient dem Trennen von grossen XML-Datenmengen in mehrere Dateien. Diese Art der Datenaufbereitung ermöglicht es sowohl beim Export, als auch beim Import, den Speicherverbrauch zu reduzieren und mit grösseren Datenmengen arbeiten zu können. Es ist aber zu beachten, dass dadurch nicht beliebige Datenmengen verarbeitet werden können, und dass es eventuell trotzdem notwendig ist, auf einen Streaming-Ansatz auszuweichen.

Im vorliegenden Kapitel wird das Konzept des Filesplits vorgestellt, er existiert zurzeit jedoch lediglich als Ansatz und ist noch nicht im Framework umgesetzt.

Tabellen-Bäume (Exkurs)
Für das Durchführen eines Exports muss das Framework für die zu exportierenden Tabellen eine Reihenfolge festlegen. Die primären Kriterien dafür sind die Relation zwischen den Tabellen.

Grundsätzlich gilt: PrimaryKey-Tabelle vor ForeignKey-Tabelle

Die Reihenfolge von nicht verknüpften Tabellen ist nicht relevant, sollte aber jeder Zeit reproduziert werden können.

Ein Verarbeitungs-Baum (Bäume) aus Tabellen (*VM10, VM70, VM71, VM72, VM97, VMSM, VMSD1, VMSD2*) könnte deshalb etwa wie folgt aussehen:
(Primär-Tabellen = Root-Knoten eines Baums - in Fettschrift)

- **VM10**
  - VM70
    - VM71
  - VM72
- **VM97**
- **VMSM**
  - VMSD1
  - VMSD2

Bei den Tabellen-Bäumen handelt es sich um die SMEEX-Archiv-Struktur, und NICHT um applikationsspezifische Daten!

Filesplit
Bei einem Filesplit gilt es unbedingt zu berücksichtigen, dass jede Datei in sich immer konsistent sein muss (d.h. es können alle Referenzen aufgelöst werden). So gesehen kann dieser Vorgang als Export im SMEEX-Export betrachtet werden. Die einzelnen Dateien werden dabei sequentiell nacheinander aufbereitet. Basierend auf den Tabellen-Bäumen, gibt es für eine Trennung zwei Möglichkeiten:

1. Auslagern von ganzen Tabellen-Bäumen (komplette Primär-Tabelle mit allen verknüpften Sekundär-Tabellen, im obigen Beispiel z.B. die Tabellen *VM10*, *VM70*, *VM71* und *VM72*). Es können sowohl ein Baum, als auch mehrere Bäume separiert werden (es sind jedoch KEINE Teilbäume möglich!)
2. Auslagern von Tabellen-Bäumen basierend auf den Datensätzen innerhalb der Primär-Tabelle (dabei wird zusätzlich zur ersten Möglichkeit eine weitere Teilung erreicht; z.B. für den Export von Patienten in separate Dateien) (z.B. eine Auswahl der Einträge aus *VMSwitchMaster* mit allen verknüpften Einträgen aus *VMSwitchDefinition* und *VMSwitchDetail*)

Es ist auch möglich, innerhalb eines Archivs beide Varianten einzusetzen.

Die Konfiguration eines Filesplits findet im Export- bzw. Import-Profil statt (Profile siehe 4.4). Die für die Konfiguration nötigen Informationen werden diesem Dokument hinzugefügt, sobald der Filesplit zur Verfügung steht.

4.2.2 Metadatenverzeichnis

Die Metadaten werden innerhalb des Files „index.xml“ abgelegt. Dieses File muss genau einmal pro Archiv vorhanden sein. Es dient als Einstiegspunkt für die Verarbeitung. Das Format entspricht demjenigen eines Datenfiles (siehe nachfolgendes Kapitel). Zusätzlich werden aber auch die möglichen Tabellen und Spalten genauer definiert.

Folgende Elemente sind innerhalb des Files erlaubt:

- tables
- columns
- relations
- general
- files

Die Elemente “tables”, “columns“ und „relations“ beschreiben die interne Struktur der vorhandenen Daten. Dabei wird keine Rücksicht auf die Anzahl Datenfiles genommen. Das enthaltene Schema entspricht der Summe aller Schemen der Datenfiles. Das Element „files“ listet alle Files im Archiv auf. Das Element „general“ wird für allgemeine Angaben verwendet. Das XML-Schema der Metadaten ist im Anhang zu finden (siehe 7.1). Im Schema sind auch die genauen Datentypen zu jedem Element mit allfälligen Maximalgrössen definiert.

4.2.2.1 tables

Informationen zu den vorhandenen Daten-Tabellen.

Elemente:

Name	Typ	Beschreibung
name	String	Name der Tabelle im Datenfile
friendlyName	String	Tabellenname zur Darstellung auf einem UI
rowCount	Integer	Anzahl Datensätze
smeexId	String	ID der Tabelle

Tabelle 1: Metadatenformat für Daten-Tabellen

Beispiel:

```
<tables>
    <name>VM70</name>
    <friendlyName>Behandlungen</friendlyName>
    <rowCount>1</rowCount>
    <smeexId>0.1.2.3.4.1943701441</smeexId>
</tables>
```

Code 1: Informationen zu Tabellen im Metadatenverzeichnis

4.2.2.2 columns

Spezifikation der Tabellen-Spalten.

Elemente:

Name	Typ	Beschreibung
table	String	Name der Tabelle
name	String	Name der Spalte im Datenfile
friendlyName	String	Spaltenname zur Darstellung auf einem UI
type	String	XSD-Typ der Spalte
Length	Integer	max. Länge der Daten (z.B. für Strings)
isPrimaryKey	Boolean	Kennzeichnen des Primary-Keys
isNullable	Boolean	Spalte darf NULL-Werte enthalten
smeexId	String	ID der Spalte

Tabelle 2: Metadatenformat für Tabellen-Spalten

Beispiel:

```
<columns>
    <table>VM70</table>
    <name>Typ</name>
    <friendlyName>Behandlungs-Typ</friendlyName>
    <type>string</type>
    <length>5</length>
    <isPrimaryKey>false</isPrimaryKey>
    <isNullable>false</isNullable>
    <smeexId>1.2.3.5.2</smeexId>
</columns>
```

Code 2: Informationen zu Tabellen-Spalten im Metadatenverzeichnis

4.2.2.3 relations

Spezifikation der Tabellen-Relationen.

Elemente:

Name	Typ	Beschreibung
primaryKeyTable	String	Name der Primary-Key-Tabelle
primaryKeyColumns	String	Name der Primary-Key-Spalten (kommasepariert)
foreignKeyTable	String	Name der Foreign-Key-Tabelle
foreignKeyColumns	String	Name der Foreign-Key-Spalten (kommasepariert)

Tabelle 3: Metadatenformat für Tabellen-Relationen

Beispiel:

```
<relations>
    <primaryKeyTable>VM10</primaryKeyTable>
    <primaryKeyColumns>PatID,DocID</primaryKeyColumns>
    <foreignKeyTable>VM70</foreignKeyTable>
    <foreignKeyColumns>Patient,Doctor</foreignKeyColumns>
</relations>
```

Code 3: Informationen zu Tabellen-Relationen im Metadatenverzeichnis

4.2.2.4 general

Spezifikation zum ganzen Archiv. Der Inhalt des Elements „general“ kann beliebig erweitert werden. Es ist zu beachten, dass Nutzdaten NICHT in diesem Element abgelegt werden dürfen, sondern innerhalb eines Datenfiles.

Elemente:

Name	Typ	Beschreibung
name	String	Name der Information
value	String	Wert

Tabelle 4: Metadatenformat für allgemeine Archiv-Informationen

Vordefinierte Einträge sind:

Name	Kardinalität	Beschreibung
author	1	Post-Anschrift des Erstellers z.B. „Dr. med. Max Muster Deisrütistrasse 10 8472 Oberohringen“
date	1	Datum und Zeit der Erstellung im Format „yyyy-MM-ddTHH:mm:ss.fffffff+01:00“ z.B. „2008-11-13T14:27+01:00“
generatorId	1	eindeutige ID (smeexID) der Applikation, die das File erstellt hat z.B. „1.2.3.4.5.6“
id	1	eindeutige Dokumenten-ID (smeexID) z.B. „1.2.3.4.5.6“
language	1	Sprache des Dokuments gem. ISO-639-1 und ISO-3166 z.B. „de-DE“, „en-US“
email	0..1	Email-Adresse des Erstellers
phone	0..1	Telefon-Nummer des Erstellers
templateId	0..1	Falls für einen Datenaustausch zusätzliche Bestimmungen gelten sollen, können diese über separate Templates definiert werden. Der entsprechende Name kann hier eingetragen werden. Es ist zu beachten, dass Templates nur zusätzliche Einschränkungen definieren können (z.B. zu verwendende Tabellennamen in den Datenfiles) und dass die resultierenden Daten immer kompatibel zum Standard sind!
templateUrl	0..1	Wenn ein Template angegeben wurde, kann hier die URL zur Dokumentation des Templates angegeben werden.
title	0..1	Titel zur Darstellung auf einem UI

Tabelle 5: Vordefinierte Einträge für allgemeine Archiv-Informationen

Beispiel:

```
<general>
    <name>id</name>
    <value>1.2.3.4.5.6</value>
</general>
```

Code 4: Allgemeine Informationen zum SMEEX Archiv im Metadatenverzeichnis

4.2.2.5 files

Spezifikation der vorhandenen Files:

Elemente:

Name	Typ	Beschreibung
name	String	Name des Files (inkl. Erweiterung)
friendlyName	String	optionaler Kommentar für das File
type	String	Typ des Files Mögliche Wert: “index”, “data”, “annex“
size	Integer	Grösse des unkomprimierten Files in Bytes (index.xml wird immer mit Grösse 0 angegeben!)

Tabelle 6: Metadatenformat für Datenfile-Informationen

Beispiel:

```
<files>
    <name>dataset.xml</name>
    <friendlyName>Daten von Patient 21</friendlyName>
    <type>data</type>
    <size>622</size>
</files>
```

Code 5: Informationen zu den vorhandenen FIles im Metadatenverzeichnis

4.2.2.6 Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<smeex xmlns="http://www.vitodata.ch/xmlns/smeex/1">
  <general>
    <name>author</name>
    <value>
      Dr. med. Max Muster
      Deisrütistrasse 10
      8472 Oberohringen
    </value>
  </general>
  <general>
    <name>date</name>
    <value>2008-11-13T14:27+01:00</value>
  </general>
  <general>
    <name>generatorId</name>
    <value>1.2.3.4.9.1</value>
  </general>
  <general>
    <name>id</name>
    <value>1.2.3.4.10.1</value>
  </general>
  <general>
    <name>language</name>
    <value>de-CH</value>
  </general>
  <files>
    <name>index.xml</name>
    <friendlyName>Metadaten</friendlyName>
    <type>index</type>
    <size>0</size>
  </files>
  <files>
    <name>dataset.xml</name>
    <friendlyName>Daten-Export</friendlyName>
    <type>data</type>
    <size>622</size>
  </files>
  <tables>
    <name>vm10</name>
    <friendlyName>Patienten-Stamm</friendlyName>
    <rowCount>3</rowCount>
    <smeexId>0.1.2.3.4.1544696801</smeexId>
  </tables>
  <tables>
    <name>vm70</name>
    <friendlyName>Behandlungen</friendlyName>
    <rowCount>1</rowCount>
    <smeexId>0.1.2.3.4.1943701441</smeexId>
  </tables>
```

```
<columns>
  <table>vm10</table>
  <name>id</name>
  <friendlyName>Patienten-Nummer</friendlyName>
  <type>int</type>
  <length>0</length>
  <isPrimaryKey>true</isPrimaryKey>
  <isNullable>false</isNullable>
  <smeexId>1.2.3.4.1</smeexId>
</columns>
<columns>
  <table>vm10</table>
  <name>name</name>
  <friendlyName>Patienten-Name</friendlyName>
  <type>string</type>
  <length>128</length>
  <isPrimaryKey>false</isPrimaryKey>
  <isNullable>false</isNullable>
  <smeexId>1.2.3.4.2</smeexId>
</columns>
<columns>
  <table>vm10</table>
  <name>date</name>
  <friendlyName>Geburtstag</friendlyName>
  <type>datetime</type>
  <length>0</length>
  <isPrimaryKey>false</isPrimaryKey>
  <isNullable>true</isNullable>
  <smeexId>1.2.3.4.3</smeexId>
</columns>
<columns>
  <table>vm10</table>
  <name>decimal</name>
  <friendlyName>Wert</friendlyName>
  <type>decimal</type>
  <length>0</length>
  <isPrimaryKey>false</isPrimaryKey>
  <isNullable>true</isNullable>
  <smeexId>1.2.3.4.4</smeexId>
</columns>
<columns>
  <table>vm70</table>
  <name>id</name>
  <friendlyName>Behandlungs-Nummer</friendlyName>
  <type>int</type>
  <length>0</length>
  <isPrimaryKey>true</isPrimaryKey>
  <isNullable>false</isNullable>
  <smeexId>1.2.3.5.1</smeexId>
</columns>
<columns>
  <table>vm70</table>
  <name>patient</name>
  <friendlyName>Patienten-Nummer</friendlyName>
  <type>int</type>
  <length>0</length>
  <isPrimaryKey>true</isPrimaryKey>
  <isNullable>false</isNullable>
  <smeexId>1.2.3.4.1</smeexId>
</columns>
```

```
    <columns>
      <table>vm70</table>
      <name>typ</name>
      <friendlyName>Behandlungs-Typ</friendlyName>
      <type>string</type>
      <length>5</length>
      <isPrimaryKey>false</isPrimaryKey>
      <isNullable>false</isNullable>
      <smeexId>1.2.3.5.2</smeexId>
    </columns>
    <relations>
      <primaryKeyTable>vm10</primaryKeyTable>
      <primaryKeyColumns>id</primaryKeyColumns>
      <foreignKeyTable>vm70</foreignKeyTable>
      <foreignKeyColumns>patient</foreignKeyColumns>
    </relations>
  </smeex>
```

Code 6: Metadatenverzeichnis im SMEEX-Archiv

4.2.3 XML-Datenfiles

Die Datenfiles befinden sich innerhalb des Archivs in einem Unterordner „data“.
Sie basieren auf einer flachen Struktur (analog den Metadaten). Die Daten werden innerhalb der Struktur als Text-Nodes (nicht als Attribute) eingefügt. Wenn innerhalb eines SMEEX-Archivs mehrere Datenfiles verwendet werden, darf derselbe Datensatz nur in einem File vorhanden sein!
Die Vorgabe eines XML-Schemas für die Datenfiles ist wegen ihrer dynamischen Struktur nicht möglich. Wird ein Schema benötigt, muss es für jedes Datenfile aus den Metadaten erzeugt werden.

4.2.3.1 Struktur

Im XML werden die folgenden drei Tag-Ebenen verwendet:

```
<ROOT><TABLE><COLUMN>WERT</COLUMN></TABLE></ROOT>
```

Beispiel:

```
<smeex><vm10><pa_nr>1</pa_nr></vm10></smeex>
```

Die Tags/Nodes haben dabei die folgende Bedeutung:

- **ROOT**
  Root-Node des Dokuments; zwingend „<smeex>“
- **TABLE**
  Zeile der Tabelle „TABLE“. Daraus folgt, dass für eine Tabelle kein separater Node erstellt wird, sondern alle Zeilen einer Tabelle denselben Tag-Namen verwenden.
- **COLUMN**
  Spalten-Name des Werts
- „**WERT**“
  Effektiver Daten-Wert

4.2.3.2 Namespace

Der für die Entwicklung des Standards verwendete Namespace lautet:

http://www.smeex.ch/xmlns/smeex/1

4.2.3.3 Zahlen

Nummerische Werte werden mit nachfolgend aufgeführten Format abgebildet (dabei ist wichtig, dass KEIN Tausender-Trennzeichen verwendet wird, und dass der Ganzzahl-Anteil durch einen Punkt von den Nachkommastellen getrennt ist):

Format:

###0.##

Beispiele:

- 1
- 1000
- 23.05

4.2.3.4 Datum / Zeit

Datum / Zeit-Werte werden mit folgendem Format abgebildet:

yyyy-MM-dd	T	HH:mm:ss.fffffff	+01:00
Datum		Zeit	Zeitzone

yyyy	Vierstellige Jahreszahl
MM	Zweistelliger Monat
dd	Zweistelliger Tag im Monat
T	Konstante für Zeit-Beginn
HH	Stunde im 24h-Format
mm	Minuten
ss	Sekunden
fffffff	Optional – Sekundenbruchteil
+01:00	Zeitverschiebung zur GMT (entspricht Zeitzone)

Tabelle 7: Datum und Zeit Formate in SMEEX Datenfiles

Beispiel:

2008-11-13T14:27:43.3423128+01:00

4.2.3.5 Beispiel

Folgendes Beispiel beinhaltet zwei Tabellen („vm10“ und „vm70“) und vier Datenzeilen (3 x vm10, 1 x vm70):

```
<smeex xmlns="http://www.vitodata.ch/xmlns/smeex/1">
  <vm10>
    <id>1</id>
    <name>Entry1</name>
    <date>2008-11-14T09:37:10.1312639+01:00</date>
    <decimal>10000.01</decimal>
  </vm10>
  <vm10>
    <id>2</id>
    <name>Entry2</name>
    <date>2008-11-14T09:37:10.1468639+01:00</date>
    <decimal>20000.02</decimal>
  </vm10>
  <vm10>
    <id>3</id>
    <name>Entry3</name>
    <date>2008-11-14T09:37:10.1468639+01:00</date>
    <decimal>30000.03</decimal>
  </vm10>
  <vm70>
    <id>1</id>
    <patient>1</patient>
    <typ>STAT</typ>
  </vm70>
</smeex>
```

Code 7: Inhalt von Datenfiles im SMEEX Archiv

4.2.4 Binäre Datenfiles

Die binären Datenfiles (Anhangfiles) befinden sich innerhalb des Archivs in einem Unterordner „annex“. Der Ordner ist optional, gibt es keine solchen Files, muss auch der Ordner nicht vorhanden sein.
Das Format von Anhangfiles kann beliebig gewählt werden (Binär oder Text). Für die Verarbeitung dieser Files ist die File-erzeugende Applikation verantwortlich. Es ist nach Möglichkeit darauf zu achten, dass gängige Formate (z.B. PDF, XPS, DOC, TXT, PNG, JPG…) gewählt werden. Obschon der Standard diese Files nicht „auswertet“ (d.h. verarbeitet und auf Validität prüft), sollte die importierende Applikation die Dokumente darstellen können. Zwingend zu beachten ist, dass die File-Erweiterung des Anhangs mit dem Inhalt des Files übereinstimmt; innerhalb des Files „test.jpg“ soll auch ein JPG-Bild vorhanden sein und kein BMP!

4.3 Das SMEEX Framework

4.3.1 Bezugsquelle

Das SMEEX Framework inklusive Dokumentation und Demo-Applikation steht für VSFM Mitglieder im Mitgliederbereich der Homepage des SMEEX Projekts (http://www.smeex.ch) zur Verfügung. Die Konditionen der Nutzung werden auf der Homepage angegeben.
Das vorliegende Dokument ist für allen interessierte Personen und Gruppierungen frei zugänglich, eine VSFM Mitgliedschaft wird nicht vorausgesetzt.

4.3.2 Umfang

Im Mitgliederbereich der VSFM Homepage wird Folgendes zur Verfügung gestellt:

Technische Dokumentation
Das vorliegende Dokument.

SMEEX Framework
.NET Bibliothek, die die Schnittstellen Definition des SMEEX Frameworks sowie eine Implementation der Schnittstelle enthält. Die Nutzung der Implementation ist optional, es steht jedem Mitglied frei, eine eigene Implementation zu erstellen und zu nutzen.

Hilfe-File zum SMEEX Framework
Windows-Hilfe File (*.chm) mit der technischen Dokumentation des Frameworks und der zur Verfügung gestellten Standard Implementation.

smeexIDManager
Tool zur Verwaltung der smeexIDs. Es enthält alle im SMEEX Standard definierten Datenfelder inklusive Datentyp, Pattern und Kardinalität. Der smeexIDManager bezieht seine Daten aus der enthaltenen MS Access 2007 Datenbank.
Details siehe Kapitel 4.6.3.

smerfIDManager
Tool zur Verwaltung des SMERF. Es enthält alle im SMERF definierten Knoten inklusive UCUM Einheit und äquivalenten LOINC Codes. Der smerfIDManager bezieht seine Daten aus der enthaltenen MS Access 2007 Datenbank.
Details siehe Kapitel 4.7.4

Demo Applikation
Demo Applikation inklusive Sourcecode, die die Nutzung des SMEEX Framework demonstriert. Die Demo Applikation bezieht ihre Daten aus einer MS Access 2007 Datenbank, die ebenfalls enthalten ist.
Details siehe Kapitel 4.8.

UCUM Bibliothek + UCUM Demo Applikation
Auf dem UCUM Standard basierende .NET Bibliothek für die Konvertierung und Validierung von Masseinheiten.

Demo Applikation inklusive Sourcecode, die die Nutzung der UCUM Bibliothek demonstriert. In der Demo Applikation können Einheiten validiert und ineinander konvertiert werden.

4.3.3 Funktionsumfang

Das SMEEX Framework stellt Schnittstellen für folgende Funktionalität zur Verfügung:

- Anzeige von SMEEX-Archiven
- Export von Daten in SMEEX Archive
- Import von Daten aus SMEEX Archiven
- Nutzung von PlugIns, darunter Kommunikatoren, Transformatoren, spezielle Import- und Export-PlugIns

Hinweis:
PlugIns stehen derzeit nicht zur Verfügung. Beachten Sie bitte entsprechende Hinweise auf der Webseite.

4.3.4 Schnittstellen

Die Schnittstellen für das SMEEX Framework sind im Namespace **Vsfm.Smeex.Framework.Core.Interfaces** definiert. Eine detaillierte Auflistung inklusive aller Member ist im ebenfalls zur Verfügung stehenden Windows Hilfe File (*.chm) zu finden (siehe 7.3).

4.3.4.1 UI Schnittstellen

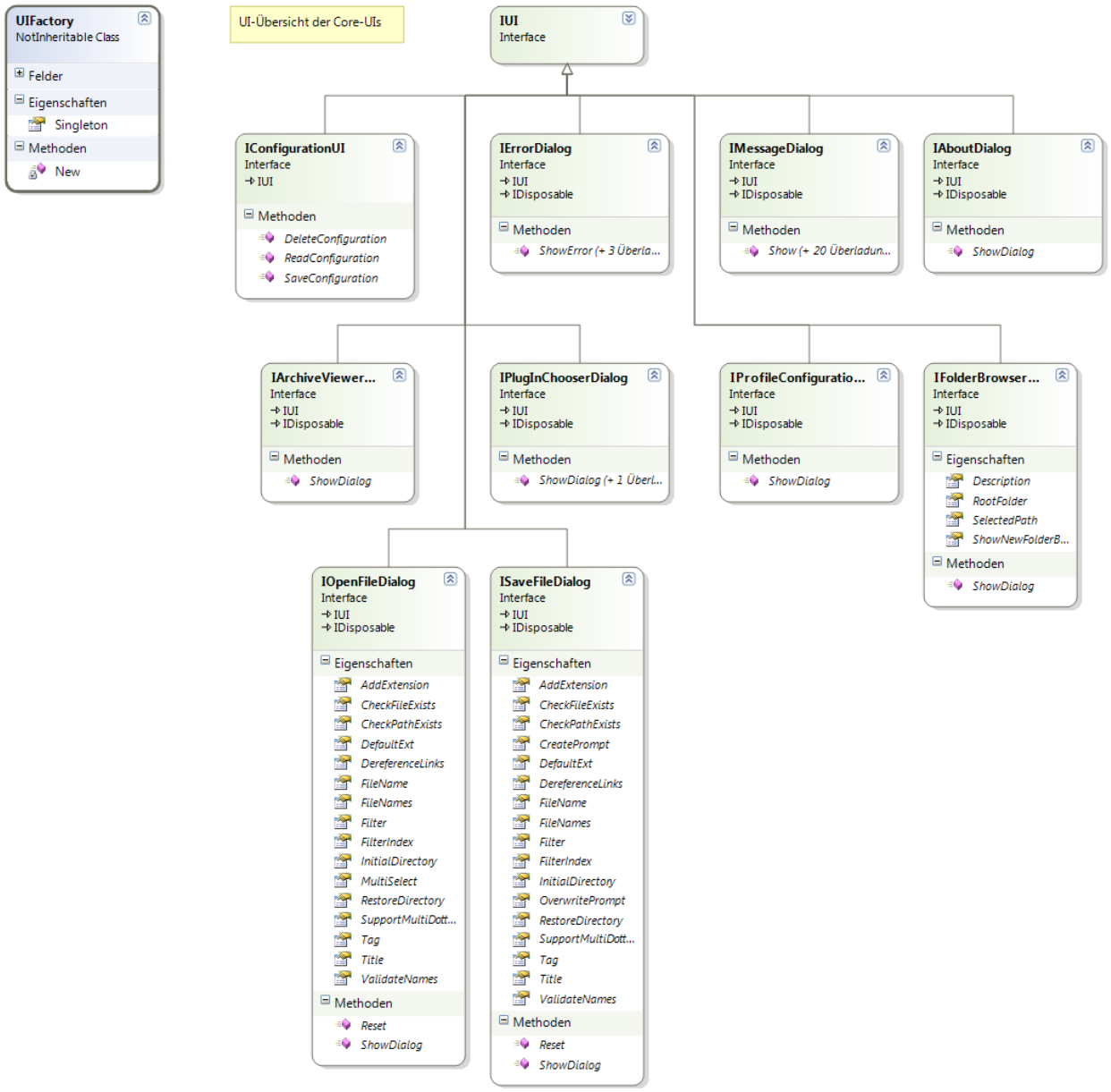

Abbildung 5: Übersicht der Core-UI Schnittstellen im SMEEX Framework

Schnittstelle	Beschreibung
IAboutDialog	Interface für den allgemeinen About-Dialog
IArchiveViewerDialog	Interface für die Anzeige von SMEEX-Archiven
IConfigurationUI	Interface für die Bearbeitung von PlugIn Konfigurationen
IErrorDialog	Interface für die Darstellung von Fehlermeldungen
IFolderBrowserDialog	Interface für einen Verzeichnis-Auswahldialog
IMessageDialog	Interface für die Darstellung von allgemeinen Meldungen
IOpenFileDialog	
IPlugInChooserDialog	UI zur Auswahl eines einzelnen PlugIns aus einer Liste
IProfileConfigurationDialog	Interface für die Bearbeitung von Profile-Konfigurationen

Schnittstelle	Beschreibung
ISaveFileDialog	
IUI	Basis-Interface für alle UI-Elemente, die über die *UIFactory* verwaltet werden sollen
IUIFactory	Interface für UI-Factories

Tabelle 8: Liste der UI-Schnittstellen im SMEEX Framework

4.3.4.2 PlugIn Schnittstellen

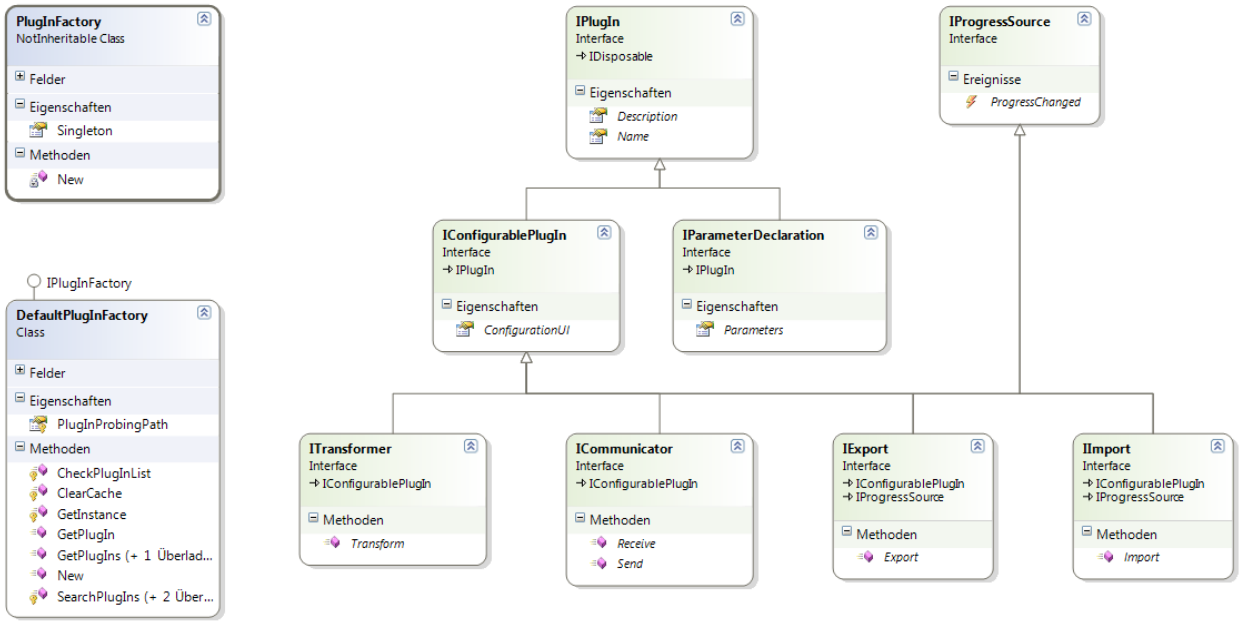

Abbildung 6: Übersicht der PlugIn Schnittstellen im SMEEX Framework

Schnittstelle	Beschreibung
ICommunicator	Basis-Interface für alle SMEEX-Kommunikatoren
IConfigurablePlugIn	Interface für konfigurierbare PlugIns
IDataDesigner	Interface für Plug.-Ins, die einen Daten-Designer bereitstellen möchten. Es können mehrere Daten-Designer parallel im SMEEX-Framework vorhanden sein, weshalb dieses Interface von *IPlugIn* und nicht von *IUI* ableitet. Beim Starten des Designers soll der Benutzer auswählen können, welchen Designer er verwenden soll. Die im Designer verwendeten UIs sollen wo möglich und sinnvoll wieder über *IUI*, resp. die *UIFactory* bereitgestellt werden. Als Spezial-Fall des Designers kann auch ein Import aus einer anderen Quelle realisiert werden.
IExport	Basis-Interface für SMEEX-Export-PlugIns
IImport	Basis-Interface für SMEEX-Import-PlugIns
IParameter	Basis-Interface für PlugIn-Parameter (siehe auch 4.3.4.3)

Schnittstelle	Beschreibung
IParameterDeclaration	Basis-Interface für PlugIns, die Parameter deklarieren. Parameter werden nicht von allen PlugIns verwendet. Die Implementation dieses Interfaces ist deshalb optional und wird durch den PlugIn-Entwickler definiert. Bei der Verwendung von Parametern sollen diese gegenüber der aufrufenden Applikation sichtbar gemacht werden, damit diese falls möglich entsprechende Werte beim Aufruf übergeben kann. Es ist möglich, dass sich mehrere PlugIns denselben Parameter teilen, resp. diesen mehrfach definieren. Dies stellt für das Framework kein Problem dar, jedoch ist darauf zu achten, dass der Name des Parameters (*IParameter.Name*) immer gleich geschrieben wird! Die durch dieses Interface definierten Parameter werden über die Klasse *ProfileConfiguration* global zur Verfügung gestellt (*SupportedParameters*).
IPlugIn	Basis-Interface für alle SMEEX-PlugIns
IPlugInFactory	Interface für PlugIn-Factories
IProgressSource	Basis-Interface für die Anzeige von Fortschritts-Anzeigen
ITransformer	Basis-Interface für alle SMEEX-Transformatoren

Tabelle 9: Liste der PlugIn Schnittstellen im SMEEX Framework

4.3.4.3 Sonstige Schnittstellen

Abbildung 7: Übersicht der sonstigen Schnittstellen im SMEEX Framework

Interface	Description
IFrameworkEvents	definiert die im SMEEX Framework ausgelösten Events
IParameter	Basis-Interface für PlugIn-Parameter
IStorage	Interface für den Applikations-spezifischen Zugriff auf die Daten

Tabelle 10: Liste der sonstigen Schnittstellen im SMEEX Framework

4.3.4.4 Exceptions

Das Exceptions-Handling bei Ex- und Import hängt von der konkreten Implementierung der genutzten PlugIns ab. Allgemein zur Verfügung stehende Exceptions sind mit einer Ausnahme im Framework nicht deklariert.

Exception	Auslösegrund
UIRegistrationException	Eine nicht registrierte UI wird bei der *UIFactory* angefordert.

Tabelle 11: Im Framework nach aussen sichtbare Exceptions

Nach aussen sichtbare Exceptions für das Archiv-Handling siehe 4.3.7.1.

4.3.5 Nutzung

4.3.5.1 Bibliotheken des Frameworks

Die Namen der Bibliotheken entsprechen dem Namespace ihres jeweiligen Inhalts. Beispiel: *Vsfm.Smeex.Framework.Core.dll* enthält alle Objekte in diesem Namespace.

Bibliothek	Inhalt
Vsfm.Smeex.DemoDB.Integration.dll	Demo Applikation
Vsfm.Smeex.Framework.Common.dll	Implementierung des SMEEX Frameworks, Archiv-Handling
Vsfm.Smeex.Framework.Core.dll	Eigentliches SMEEX Framework mit Schnittstellen und Einstiegsklasse (siehe 4.3.5.2)
Vsfm.Smeex.Framework.Core.UI.dll	UI-Implementierungen für die Core-Bibliothek
Vsfm.Smeex.Framework.PlugIn.dll	Implementierungen von PlugIns
Vsfm.Smeex.Framework.Zip.dll	Handling von ZIP-Archiven

Tabelle 12: Im SMEEX Framework enthaltene Klassen-Bibliotheken

4.3.5.2 Einstiegspunkt in das Framework

Der Einstiegspunkt in das SMEEX Framework ist die Klasse *SmeexFramework* im Namespace **Vsfm.Smeex.Framework.Core**. Sie stellt die Basisfunktionalität des SmeexFrameworks zur Verfügung. Falls ein anderes Verhalten benötigt wird, wird empfohlen, diese Klasse zu erweitern und die betreffenden Methoden zu überschreiben.
Die Klasse *SmeexFramework* stellt lediglich ein Gerüst dar, das die Infrastruktur für spezifische PlugIns bietet. Bei der Instanziierung muss ihr ein Objekt vom Typ *IStorage* übergeben werden, das für den Applikations-spezifischen Zugriff auf die zu verarbeitenden Daten zuständig ist.

```
Public Sub New(ByVal storage As IStorage)
```
Code 8: Konstruktor der Klasse SmeexFramework

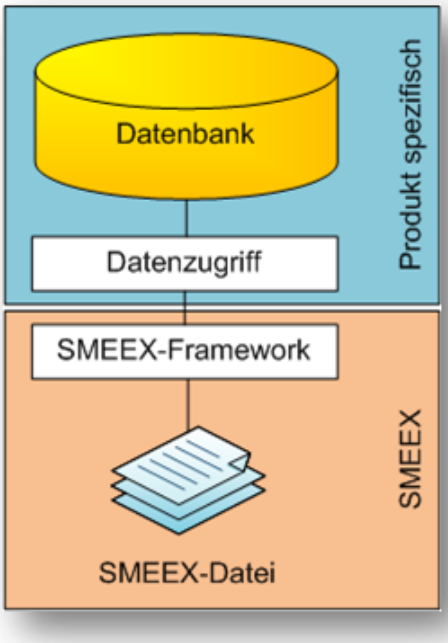

Abbildung 8: Einbettung SMEEX Framework: Komponenten Struktur

Bei Aufruf der Methoden für Export und Import muss ein Profil übergeben werden. Ist in dem Profil z.B. kein PlugIn für den Export definiert, wird das Standard-Export-PlugIn genommen, das im Framework enthalten ist. Gleiches gilt für den Import.
Besonders für die Verarbeitung grosser Datenmengen wird empfohlen, nicht das Standard-PlugIn, sondern ein eigens dafür entwickeltes PlugIn zu verwenden.

```
Public Overridable Function Export(ByVal owner As IWin32Window, ByVal profile
As ProfileConfiguration) As Boolean
```
Code 9: Signatur der ExportMethode in der Klasse SmeexFramework

```
Public Overridable Function Import(ByVal owner As IWin32Window, ByVal profile
As ProfileConfiguration) As Boolean
```
Code 10: Signatur der Import Methode in der Klasse SmeexFramework

Soll nach dem Export das resultierende SMEEX Archiv in ein Dritt-Format wie z.B. CDA transformiert werden, kann ein oder mehrere im Profil definierten Transformator PlugIns aufgerufen werden. Zurzeit gibt es dafür keine Standard-Implementierung im SMEEX Framework.

```
Protected Overridable Function TransformExport(ByVal profile As
ProfileConfiguration, ByVal arch As Archive) As String
```
Code 11: Signatur der TransformExport Methode in der Klasse SmeexFramework

Analog kann auch vor dem Import eine Transformation aus einem anderen Format in den SMEEX Standard durchgeführt werden.

```
Protected Overridable Function TransformImport(ByVal profile As
ProfileConfiguration, ByVal input As String) As Archive
```
Code 12: Signatur der Methode TransformImport in der Klasse SmeexFramework

Für die Kommunikation der Quell- oder der Zieldaten können im Profil Kommunikatoren definiert werden. Damit können zum einen nach dem abgeschlossenen Export die exportierten Daten (SMEEX Archiv oder im Fremdformat nach Transformation) zu einem bestimmten Ziel übertragen werden, zum anderen die Quelldaten vor einem Import „abgeholt“ werden. Sind mehrere Kommunikatoren im Profil hinterlegt, werden sie innerhalb der Methode nacheinander aufgerufen.

```
Protected Overridable Sub CommunicateExport(ByVal profile As
ProfileConfiguration, ByVal arch As Archive, ByVal output As String)
```
Code 13: Signatur der Methode CommunicateExport in der Klasse SmeexFramework

```
Protected Overridable Function CommunicateImport(ByVal profile As
ProfileConfiguration) As String
```
Code 14: Signatur der Methode CommunicateImport in der Klasse SmeexFramework

Im Framework ist der Kommunikator *FileCommunicator* enthalten. Er ist aber nicht als Standard hinterlegt, sondern muss explizit dem verwendeten Profil zugewiesen werden. Er ist im Namespace **Vsfm.Smeex.Framework.PlugIn** beheimatet.
Pfad und Filename des zu schreibenden oder lesenden Files werden hierbei als PlugIn Parameter in der Profil-Konfiguration hinterlegt (siehe 4.5.5)

Ein Beispiel für die Nutzung des SMEEX Frameworks ist in der Demo Applikation (siehe 4.8) zu finden, die inklusive Quellcode zur Verfügung steht.

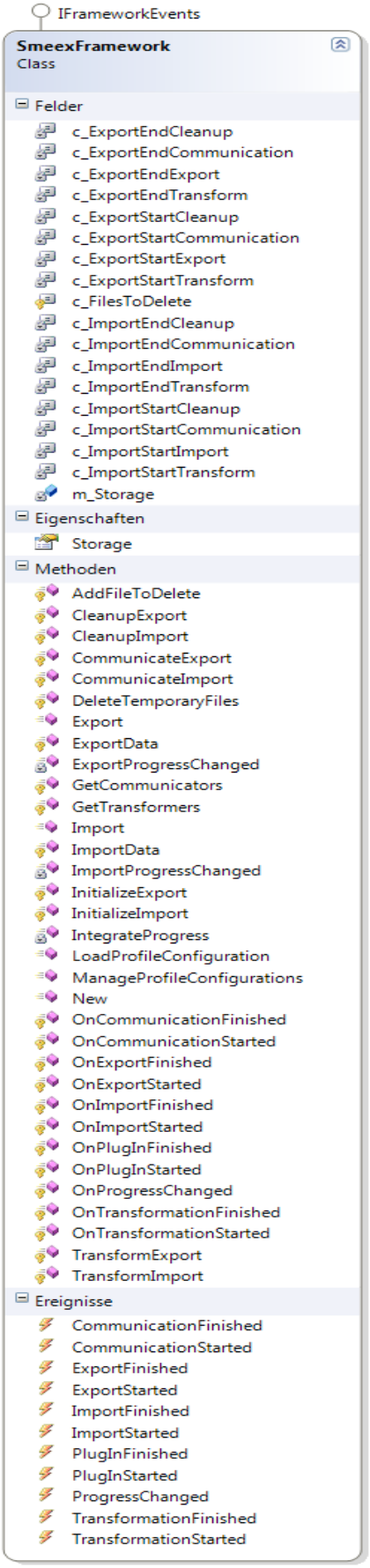

Abbildung 9: SMEEX Framework: Klasse SmeexFramework

4.3.6 Implementierung von PlugIns

Im vorliegenden Dokument ist der aktuelle Stand bzgl. PlugIns dokumentiert. Da das Framework sich aktuell im „Beta-Stadium“ befindet, können sich noch Änderungen ergeben.

Das SMEEX-Framework stellt eine PlugIn Architektur zur Verfügung, die von den unterschiedlichsten Herstellern genutzt werden kann, um die Funktionalität des Frameworks zu erweitern und in bestehende Prozesse einzugreifen.

4.3.6.1 PlugIn Architektur

Die PlugIn Architektur bietet im Wesentlichen folgende zwei Komponenten (siehe Abbildung 6):

- IPlugInFactory
- IPlugIn

IPlugInFactory

Eine *PlugInFactory* ermittelt und verwaltet die für das Framework verfügbaren PlugIns. Jede Factory muss dafür das Interface *IPlugInFactory* implementieren. Die aktive Fabrik wird mit Hilfe der Klasse *PlugInFactory* als Singleton zur Verfügung gestellt.

Achtung:
Jeder Zugriff muss zwingend über PlugInFactory.Singleton erfolgen, damit die Funktion des Frameworks nicht beeinträchtigt wird.
Das Cachen der Factory in einer privaten Variable oder ähnlichem entspricht deshalb nicht den Architekturvorgaben!

Hinweis:
Das SMEEX-Framework wird bereits mit einer entsprechenden Factory ausgeliefert, sodass das Ersetzen oder Anpassen dieser Fabrik nur in Ausnahmefällen in Betracht gezogen werden sollte.

IPlugIn

PlugIns können die verschiedensten Aufgaben innerhalb des Frameworks übernehmen. Die Gemeinsamkeit besteht darin, dass sie das Interface *IPlugIn* implementieren. Dies ist notwendig, damit die *PlugInFactory* die PlugIn Klassen als solche erkennen und zur Verfügung stellen kann.
PlugIns werden vor allem in den Bereichen "Transformation" und "Kommunikation" verwendet. Die eigens dafür zur Verfügung gestellten Interfaces *ITransformer* und *ICommunicator* dienen dabei als Basis zur Entwicklung von eigenen Erweiterungen. Beispielsweise könnte mit Hilfe des Interfaces *ICommunicator* das Framework um die Möglichkeit erweitert werden, mit FTP-Servern zu kommunizieren.

4.3.6.2 PlugIn Entwicklung

Das Entwickeln von eigenen PlugIns ist so einfach wie möglich gehalten. Eine Implementierung kann im Wesentlichen mit folgenden Schritten erfolgen:

1. Erstellen eines neuen Projekts/Assembly
2. Erstellen einer neuen Klasse, die das Interface *IPlugIn* oder eine Ableitung davon implementiert
3. Signieren der Assembly mit einem starken Namen (strong name)
4. Kopieren der Assembly (und aller Abhängigkeiten) in das Framework-Verzeichnis

Hinweise:
Für eigene UIs im PlugIn, wird auf das Kapitel zur "UI-Entwicklung" verwiesen.
Damit auch nach der Entwicklung des PlugIns zur Laufzeit erweiterte Informationen zur Verfügung stehen, sollten PlugIn-Entwickler den Einsatz der LogFaçade (siehe 4.3.7.2) in Betracht ziehen.
Falls für eine Erweiterung auch ein PlugIn Mechanismus vorgesehen ist, kann dafür auch die Infrastruktur des Frameworks verwendet werden.

4.3.6.3 UI Entwicklung

Das SMEEX-Framework wurde sowohl für den Client-Server-Betrieb, als auch für die Integration in Windows-Dienste entwickelt. Diese technischen Rahmenbedingungen stellen unterschiedliche Anforderungen an das Benutzeroberflächen-Konzept des Frameworks, als auch an allfällige Erweiterungen von Drittherstellern (siehe 4.3.6.2).
Um beide Anforderungsbereiche abdecken zu können, stehen im Framework zwei unterschiedliche Varianten von "UI-Definitionen" zur Verfügung. Zudem ist es möglich, jede dieser Oberflächen durch eigene Implementationen zu ersetzen, resp. PlugIn spezifische GUIs zu registrieren.

UI-Definitionen

Die konkret von der Applikation verwendete UI-Definition wird über die Klasse *UIFactory* festgelegt. Falls keine explizite *IUIFactory* gesetzt wird, entscheidet das Framework selbständig, ob UIs verwendet werden können, oder ob sich der aktuelle Prozess im sogenannten "headless"-Modus befindet (wodurch UIs nicht verwendet werden dürfen). Entsprechend wird eine der untenstehenden Definitionen ausgewählt.

Achtung:
Bei der Implementierung einer eigenen UIFactory, sollte mit äusserster Sorgfalt vorgegangen werden; durch eine fehlerhafte Implementation kann das gesamte Framework beeinträchtigt werden!

Um eine spezifische Implementation zu erzwingen, setzen Sie das *UIFactory.Singleton*-Property auf die gewünschte Factory-Instanz. Das Rücksetzen auf die NullUI erlaubt dem Framework wieder, die Auswahl selbst zu treffen.

```
UIFactory.Singleton = New NullUI()
```

Code 15: Rücksetzen des UIFactory.Singleton Property

Achtung:
Eine andere Factory sollte nur dann gesetzt werden, wenn dies zwingend erforderlich ist! Durch eine „falsche“ Implementierung kann das ganze Framework beeinträchtigt werden!

DefaultUI
Die Default-UI Implementation (*Vsfm.Smeex.Framework.Core.UI.DefaultUI.Factory*) stellt die Standard-Implementation für den Client-Server-Betrieb dar. Alle verwendeten UIs müssen hier zur Verfügung gestellt werden!

Wichtiger Hinweis:
Es muss sichergestellt werden, dass eigene Implementationen von IUIFactory über einen Fallback auf das DefaultUI, resp. NullUI verfügen, damit PlugIns von Drittherstellern weiterhin funktionsfähig sind!

Obwohl nicht zwingend erforderlich, wird dringend empfohlen, eigene UIs mit Hilfe von Patterns (wie z.B. MVC oder MVVM) zu entwickeln. Damit wird die Austauschbarkeit der Benutzeroberfläche erheblich erleichtert!

NullUI
Die NullUI Implementation (*Vsfm.Smeex.Framework.Core.UI.NullUI.Factory*) stellt die Standard-Implementation für den Windows-Dienst-Betrieb dar. Wie der Name bereits vermuten lässt, stellt diese Factory lediglich leere Hüllen von UIs dar, sodass der Framework- (oder PlugIn-) Code nicht unterscheiden muss. Falls z.B. innerhalb des Codes eine MessageBox (via *UIFactory*!) angefordert wird, wird bei dieser Implementation die Darstellung unterdrückt, und sofort das Resultat (Default-Button) zurückgegeben.
Nicht jedes UI, das innerhalb von *DefaultUI* zur Verfügung gestellt wird, verfügt auch über eine NullUI-Implemenation. So werden z.B. UIs, die zur Konfiguration des PlugIns verwendet werden, im *NullUI* nicht registriert (da für den Windows-Dienst-Betrieb nicht notwendig). Sollte ein entsprechendes UI trotzdem angefordert werden, wird eine Exception vom Typ *UIRegistrationException* ausgelöst.

Wichtiger Hinweis:
Es muss sichergestellt werden, dass eigene Implementationen von IUIFactory über einen Fallback auf das DefaultUI, resp. NullUI verfügen, damit PlugIns von Drittherstellern weiterhin funktionsfähig sind!

4.3.6.4 Umgang mit UIs

Jedes UI innerhalb des SMEEX-Frameworks verfügt über ein eigenes Interface, anhand dessen es identifiziert werden kann. Dieses Interface leitet immer von *IUI* ab, sodass es über *IUIFactory* verwaltet werden kann.

Für den Umgang mit UIs innerhalb des SMEEX-Frameworks, ergeben sich im Wesentlichen folgende drei Teilbereiche:

UI anfordern

Ein neues UI wird mit folgendem Aufruf angefordert (Beispiel *IMessageDialog*):

```
UIFactory.Singleton.MessageDialog.ShowDialog(...)
```

Code 16: UI-Aufruf

Oberflächen die seltener verwendet werden, resp. durch Dritthersteller dynamisch hinzugefügt werden, können über die allgemeine Methode *GetUI* angefordert werden (Beispiel *IMessageDialog*):

```
DirectCast(UIFactory.Singleton.GetUI(GetType(IMessageDialog)),
IMessageDialog).ShowDialog(...)
```

Code 17: UI-Aufruf mittels GetUI

Neue UIs hinzufügen

Für PlugIn Entwickler stellt sich die Frage, ob eigene UIs bei der Factory registriert / bezogen werden müssen. Die Antwort auf diese Frage ist **nein**, es wird jedoch empfohlen! Zum einen muss sich der PlugIn Entwickler dann nicht um den Betriebs-Modus kümmern (headless oder nicht), und zum anderen können die UIs zwecks Vereinheitlichung des optischen Erscheinungsbilds ausgetauscht werden.
In diesem Abschnitt wird das Vorgehen beschrieben, wenn die Benutzer-Oberfläche über das Framework verwaltet werden soll. Ist keine Verwaltung erwünscht, so sind "keine speziellen" Rahmenbedingungen (z.B. Interfaces) zu berücksichtigen. Folgendes Vorgehen ist für verwaltete UIs einzuhalten:

1. Erstellen eines Interfaces, das von *IUI* ableitet
2. Erstellen einer konkreten Implementation des neuen Interfaces
3. Registrieren des UIs in der DefaultUI-Factory (*RegisterUI*)
4. Registrieren des UIs in der NullUI-Factory (sofern notwendig; *RegisterUI*)

Wichtiger Hinweis:
Obwohl nicht zwingend erforderlich, wird dringend empfohlen, eigene UIs mit Hilfe von Patterns (wie z.B. MVC oder MVVM) zu entwickeln. Damit wird die Austauschbarkeit der Benutzeroberfläche erheblich erleichtert!

Bestehende UIs ersetzen
Das Ersetzen von bestehenden Benutzer-Oberflächen kann je nach Implementation des Original-GUIs sehr einfach bis sehr komplex sein! Es ist daher die Dokumentation des Original-GUIs unbedingt zu berücksichtigen.

Folgendes Vorgehen ist für das Ersetzen eines verwalteten UIs einzuhalten:

1. Erstellen einer konkreten Implementation des gewünschten Interfaces
2. Registrieren des neuen UIs in der *DefaultUI*-Factory (*ReplaceUI*)
3. Registrieren des neuen UIs in der *NullUI*-Factory (sofern notwendig; *ReplaceUI*)

Hinweis:
Es ist nicht möglich, für dasselbe Interface mehrere Implementationen zu hinterlegen. Die zuletzt registrierte Implementation wird im gesamten Framework verwendet!

Hinweis:
Wenn UIs mit Elementen ausserhalb des .NET-Frameworks ersetzt werden sollen (z.B. COM-UI), dann muss zwingend ein .NET-Wrapper erstellt werden, der das gewünschte Interface implementiert und die Aufrufe anschliessend an die externe Komponente weiterleiten kann.

4.3.7 Implementierung

Die von VSFM zur Verfügung gestellte Bibliothek beinhaltet eine Implementierung des SMEEX Frameworks. Sie ist sind im Namespace **Vsfm.Smeex.Framework.Common** enthalten.

4.3.7.1 SMEEX Archiv-Handling

Um den Zugriff auf die Daten in SMEEX Archiven zu vereinfachen, wird die Datenstruktur der Metadaten (siehe 4.2.1) 1:1 auf entsprechende Klassen abgebildet. Da zum Zeitpunkt der Implementierung nur die Strukturen der Standardklassen bekannt sind, können nur sie zur Verfügung gestellt werden. Hersteller- oder PlugIn spezifische Klassen sind nicht enthalten.

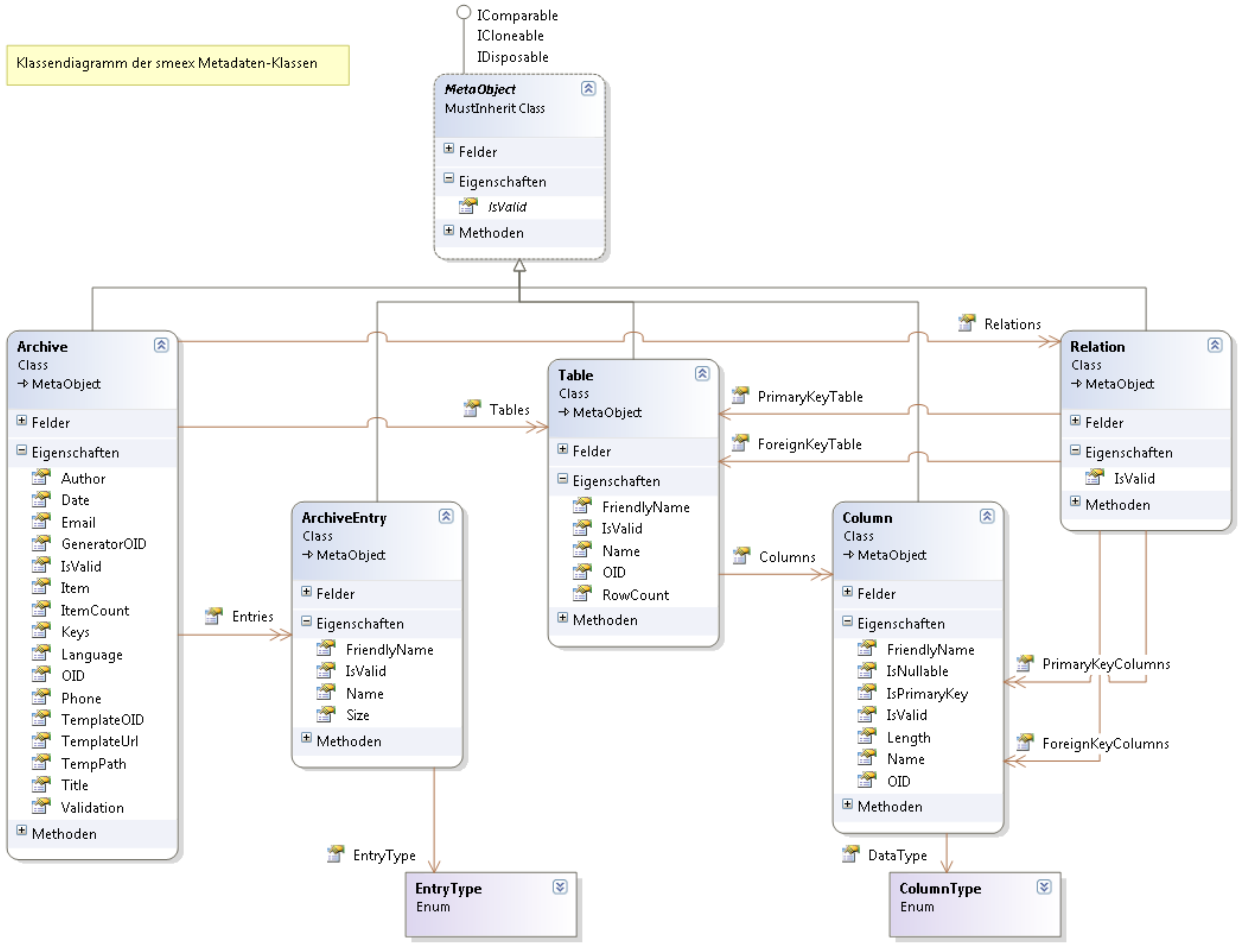

Abbildung 10: Klassendiagramm der SMEEX Metadaten-Klassen

In der Implementierung ist kein Datenbank Zugriff auf Quell- bzw. Zieldaten enthalten, sie beschränkt sich auf ein reines Daten-Handling. Der Zugriff auf die vorhandenen Daten wird ausschliesslich über eine applikationsspezifische Implementation der *IStorage*-Schnittstelle gewährleistet. Das Framework ist per se persistenzlos.

Beispiele für den Zugriff auf die Archivdaten:

Index (Metadaten)
Der Zugriff auf die allgemeinen Daten und die Struktur der XML-Daten erfolgt über die Klasse *Archive* und weitere Hilfsklassen. Verwenden Sie diese Klasse auch für das Laden und Speichern von SMEEX-Files.

```
Using a = New Archive("C:\test.smeex")
  Debug.WriteLine(a.Author)
  Debug.WriteLine(a.Item("MySpecialKey"))

  a.Item("MySecondSpecialKey") = "MySecondSpecialValue"
  a.Save()
End Using
```

Code 18: Zugriff auf Metadaten des SMEEX Archivs

XML-Daten Struktur

Die Struktur der vorhandenen XML-Daten ist über die Archiv-Instanz (*Archive*) verfügbar. Sie wird in Form von Tabellen- (*Tables*) und Spalten- (*Columns*) Objekten zur Verfügung gestellt. Das ermöglicht es, grundlegende Informationen zu den vorhandenen Daten zu ermitteln, ohne diese zu laden. Dieses Vorgehen wurde gewählt, um die Performance bei grossen Datenmengen zu optimieren.

```
Using a = New Archive("C:\test.smeex")
  For Each tbl As Table In a.Tables
    Debug.WriteLine(tbl.Name)
  Next
End Using
```

Code 19: Zugriff auf Daten-Struktur im SMEEX Archiv

Zudem stellen diese Strukturinformationen das Bindeglied zwischen den Objektidentifiern (smeexId), und den technischen Namen innerhalb der XML-Daten dar.

XML-Daten

Auf die XML-Daten kann auf mehrere Arten zugegriffen werden. Es steht sowohl ein direkter XML-Zugang, als auch eine Sicht der Daten via *DataSet* zur Verfügung.
Die entsprechenden Formate können bei der Archiv-Instanz unter Angabe des zu lesenden Files (*ArchiveEntry*) angefragt werden.

```
Dim ds As New DataSet
Using a = New Archive("C:\test.smeex")
  For Each ae As ArchiveEntry In a.Entries
    If (ae.EntryType = EntryType.Data) Then
      a.GetData(ae, ds)
      Debug.WriteLine(ds.GetXML())
    End If
  Next
End Using
```

Code 20: Zugriff auf XML-Daten im SMEEX Archiv

Binäre-Daten

Binäre Daten werden vom Framework nicht weiter verarbeitet und werden deshalb auch direkt binär wieder zur Verfügung gestellt.
Analog zu den XML-Daten können auch die Binär-Files bei der Archiv-Instanz unter Angabe des zu lesenden Files (*ArchiveEntry*) angefragt werden.

```
Dim stream As New MemoryStream()
Using a = New Archive("C:\test.smeex")
  For Each ae As ArchiveEntry In a.Entries
    If (ae.EntryType = EntryType.Annex) Then
      a.GetEntry(ae, stream)
    End If
  Next
End Using
```

Code 21: Zugriff auf binäre Daten im SMEEX Archiv

Bei der Nutzung der Archiv-Klasse können die folgenden Exceptions auftreten.

Exception	Auslösegrund
DuplicateElementException	In der Datenstruktur wird eine Spalte oder eine smeexID doppelt aufgeführt.
InvalidArchiveException	Das zu lesende File ist kein gültiges SMEEX Archiv.
ValidationException	Der Inhalt des Files entspricht keinem gültigen XML oder SMEEX Format.

Tabelle 13: Beim Archiv-Handling auftretende Exceptions

4.3.7.2 Logging

Für PlugIn Entwickler wird im Namespace **Vsfm.Smeex.Framework.Common** eine *LogFaçade* zur Verfügung gestellt.
Für die Protokollierung wird das Microsoft Logging-Framework verwendet. Das Microsoft Logging-Framework ist Bestandteil des .NET Frameworks und steht damit ohne weiteren Aufwand zur Verfügung. Technische Details können im MSDN (Microsoft Developer's Network) unter http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.logging.aspx nachgelesen werden.
Zur Konfiguration dient das Konfigurations-File der Applikation (app.config). Zudem ist es möglich zur Laufzeit weitere Listeners über die Methode Register(*TraceListener*) hinzuzufügen.

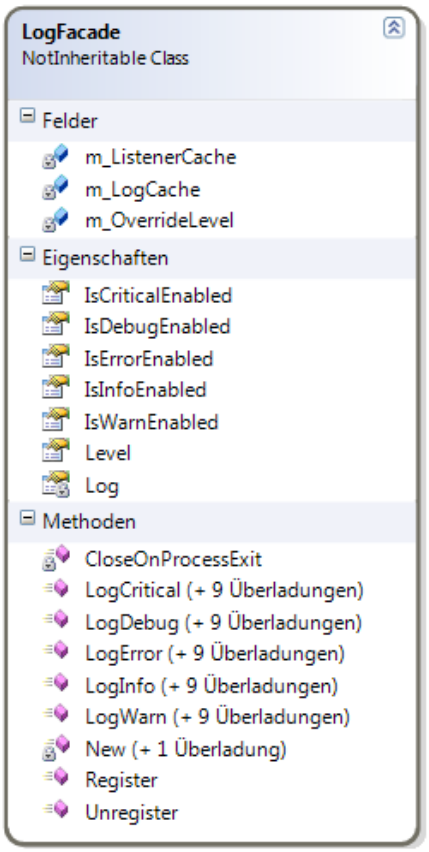

Abbildung 11: Framework Implementierung: Klasse LogFaçade

4.4 Profile

Das Datenformat des SMEEX-Standards definiert eine sehr allgemeine Sicht der abbildbaren Daten. Bei einem Export ist es jedoch nicht sinnvoll, alle zur Verfügung stehenden Daten jedes Mal zu exportieren, sondern je nach Anwendungsfall, nur gewisse Teilbereiche der Daten bereitzustellen (analoges gilt für den Import). Damit solche Anwendungsfälle definiert und genauer spezifiziert werden können, werden innerhalb des SMEEX-Frameworks Profile verwendet.

4.4.1 smeexProfile und Profile für den Datenaustausch

Im SMEEX Umfeld gibt es zwei Arten von Profilen, smeexProfile und Profile für den Datenaustausch. smeexProfile werden im Folgenden immer so bezeichnet. Wenn nur „Profil“ verwendet wird, ist ein Profil für den Datenaustausch gemeint.

smeexProfile
Ein smeexProfil definiert eine Zusammenstellung von Datenfeldern. Es besteht aus einer Liste von smeexIDs (siehe 4.6). smeexProfile dienen dazu, bestimmte Untermengen aller verfügbaren Daten zu bilden. Zum Beispiel können das alle Daten eines Patienten sein, die für eine Überweisung benötigt werden. Im Profil „fullSMEEX“ sind alle im SMEEX Standard definierten IDs enthalten. Die zur Verfügung stehenden smeexProfile sind im smeexIDManager (siehe 4.6.3) hinterlegt.
Diese Profile werden von VSFM definiert und können auf Antrag und bei generellem Bedarf durch weitere ergänzt werden.

Profile für den Datenaustausch
Profile für den Datenaustausch (kurz Profile), definieren den gesamten Ablauf eines Datenaustauschs. Im Profil für den Datenaustausch ist u.a. das für den Export zu verwendende smeexProfil enthalten.
Ein Profil beinhaltet sowohl die allgemeine anwendungsfallspezifische Konfiguration, als auch gewisse PlugInspezifischen Daten. Der allgemeine Teil des Profils soll zwischen verschiedenen Systemen ausgetauscht werden können, wogegen der PlugIn-spezifische Teil nicht generell gültig ist (nähere Informationen zu PlugIns siehe 4.5). Zu Backup- und Restore-Zwecken soll aber auch dieser Bereich exportiert und importiert werden können.
Die Profile werden in Form von XML-Daten (SMEEX-Format kompatibel) bereitgestellt. Sie werden weder komprimiert oder verschlüsselt, noch anderweitig explizit vor Manipulationen geschützt. Es wird angestrebt, den Grossteil der Kommunikation über ein Profil (fullSMEEX bzw. eine Teilmenge von fullSMEEX) zu kanalisieren, jedoch wird dies nicht in jedem Fall möglich sein.

Beispiel: Ein Profil definiert den Versand der Daten eines Patienten (smeexProfil), im Format CDA über einen FTP-Server.

4.4.2 Profil-Konfiguration

Sowohl Export- wie auch Import-Profile verfügen über ein gemeinsames Set von Konfigurations-Möglichkeiten; diese werden in diesem Kapitel beschrieben.
Die Profile sind nicht nur auf die SMEEX-Framework spezifische Konfigurationen beschränkt. Aus Benutzersicht folgt vielfach z.B. einem Export auch der Versand des exportierten Files. Es soll daher für die PlugIn-Mechanismen eine Möglichkeit bestehen, zusätzliche Konfigurationen (z.B. für einen Fileversand) im Profil zu hinterlegen.
Grundsätzlich gliedert sich damit ein Profil in folgende Konfigurationsbereiche:

- Allgemeine Konfiguration
- Daten-Struktur
- Parameter
- PlugIn-Konfiguration

4.4.2.1 Allgemeine Konfiguration

Folgende Informationen werden bereitgestellt / gespeichert:

Name	Bemerkung
Name	Darstellbarer Namen des Profils
Id	Eindeutige Identifikation des Profils
date	Erstell-Datum
author	Darstellbarer Namen des Profil-Erstellers
Ersteller smeexID	Eindeutige Identifikation des Profil-Erstellers
Description	Beschreibung / Bemerkungen zum Profil
DataStructure	Deklaration der Daten-Struktur der XML-Daten Dies umfasst sowohl die Definition der zu liefernden Tabellen und Spalten, als auch der Relationen untereinander. Format siehe Metadatenstruktur in Kapitel 4.2.1
Parameter.ParameterName	Deklaration der zu verwendenden Parameter
Transformers	Zu verwendende Transformatoren PlugIns
Communicators	Zu verwendende Kommunikatoren PlugIns
ExportPlugIn	Zu verwendende Export-Handler PlugIns
ImportPlugIn	Zu verwendende Import-Handler PlugIns
Validierung	Flag, zur Steuerung der Daten-Validierung während der Verarbeitung. Die Validierung der Metadaten kann NICHT deaktiviert werden!
Version	Version des Profil-Formats (Achtung! Nicht die Bearbeitungsversion des Profils)

Tabelle 14: Allgemeinen Konfiguration von Export und Import Profilen

Beispiel:

```
<General>
    <Key>Name</Key>
    <Value>PDF-Export</Value>
</General>
```

```
<General>
    <Key>Description</Key>
    <Value>Verwendet die Tabelle "Adressen" und exportiert die Dateien
in der Tabelle "Dateien".</Value>
</General>
<General>
    <Key>Transformers</Key>
    <Value>Vsfm.Smeex.Framework.PlugIn.Debug.FileViewer;</Value>
</General>
<General>
    <Key>ExportPlugIn</Key>
    <Value>Vsfm.Smeex.Framework.CoreTest.DefaultExportTest</Value>
</General>
<General>
    <Key>ImportPlugIn</Key>
    <Value>Vsfm.Smeex.Framework.CoreTest.DefaultImportTest</Value>
</General>
```

Code 22: Allgemeine Konfiguration von Profilen

4.4.2.2 Daten-Struktur

Die Daten-Struktur entspricht der Definition aus dem SMEEX-Format. Die Spalte „tables.rowcount“ aus der Formats-Definition wird nicht verwendet und deshalb mit dem Wert 0 belegt. Die technischen Namen (sowohl bei Tabellen als auch Spalten) werden beliebig festgelegt und zur Laufzeit überschrieben.
Grundidee dieses Ansatzes ist es, für die Erstellung eines Archivs die notwenigen Struktur-Definitionen möglichst schnell / einfach zur Verfügung zu stellen. Bei einem Import kann die erwartete Struktur abgeglichen, resp. durch das Profil die zu importierenden Daten eingeschränkt werden.
Es soll möglich sein, wenn auch nicht empfohlen, in den XML-Daten zusätzliche Daten (Tabellen und Spalten) zu liefern. Falls solche geliefert werden, sind im Archiv (nicht im Profil!) die Strukturen entsprechend zu erweitern / dokumentieren! Die Deklaration innerhalb des Profils ist daher als Minimum zu verstehen.
Falls keine Strukturdaten bereitgestellt werden, liegt es in der Verantwortung der Applikation, welche Daten verarbeitet werden sollen. Dafür kann auch der Bereich der „Hersteller-spezifische Konfiguration“ verwendet werden (siehe 4.4.2.5).

Beispiel:

```
<General>
    <Key>DataStructure</Key>
    <Value>&lt;?xml version="1.0" standalone="yes"?&gt;
&lt;smeex xmlns="http://www.smeex.ch/xmlns/smeex/1"&gt;
  &lt;general&gt;
    &lt;name&gt;author&lt;/name&gt;
    &lt;value&gt;UnitTest&lt;/value&gt;
  &lt;/general&gt;
  &lt;general&gt;
    &lt;name&gt;date&lt;/name&gt;
    &lt;value&gt;2009-08-18T20:05:07.6654042+02:00&lt;/value&gt;
  &lt;/general&gt;
  &lt;general&gt;
    &lt;name&gt;generatorId&lt;/name&gt;
    &lt;value&gt;1.2&lt;/value&gt;
  &lt;/general&gt;
```

```
&lt;general&gt;
  &lt;name&gt;id&lt;/name&gt;
  &lt;value&gt;1.2.3&lt;/value&gt;
&lt;/general&gt;
&lt;general&gt;
  &lt;name&gt;language&lt;/name&gt;
  &lt;value&gt;de&lt;/value&gt;
&lt;/general&gt;
&lt;entries&gt;
  &lt;name&gt;index.xml&lt;/name&gt;
  &lt;friendlyName&gt;index.xml&lt;/friendlyName&gt;
  &lt;type&gt;index&lt;/type&gt;
  &lt;size&gt;0&lt;/size&gt;
&lt;/entries&gt;
&lt;tables&gt;
  &lt;name&gt;T1&lt;/name&gt;
  &lt;friendlyName&gt;Profil-Konfigurationen&lt;/friendlyName&gt;
  &lt;rowCount&gt;0&lt;/rowCount&gt;
  &lt;smeexId&gt;2.16.756.5.30.1.106.1.1&lt;/smeexId&gt;
&lt;/tables&gt;
&lt;columns&gt;
  &lt;table&gt;T1&lt;/table&gt;
  &lt;name&gt;C1&lt;/name&gt;
  &lt;friendlyName&gt;Name&lt;/friendlyName&gt;
  &lt;type&gt;string&lt;/type&gt;
  &lt;length&gt;64&lt;/length&gt;
  &lt;isPrimaryKey&gt;true&lt;/isPrimaryKey&gt;
  &lt;isNullable&gt;false&lt;/isNullable&gt;
  &lt;smeexId&gt;2.16.756.5.30.1.106.1.1.1&lt;/smeexId&gt;
&lt;/columns&gt;
&lt;columns&gt;
  &lt;table&gt;T1&lt;/table&gt;
  &lt;name&gt;C2&lt;/name&gt;
  &lt;friendlyName&gt;Daten&lt;/friendlyName&gt;
  &lt;type&gt;string&lt;/type&gt;
  &lt;length&gt;65536&lt;/length&gt;
  &lt;isPrimaryKey&gt;false&lt;/isPrimaryKey&gt;
  &lt;isNullable&gt;false&lt;/isNullable&gt;
  &lt;smeexId&gt;2.16.756.5.30.1.106.1.1.2&lt;/smeexId&gt;
&lt;/columns&gt;
&lt;/smeex&gt;</Value>
</General>
```

Code 23: Deklaration von Daten-Strukturen in Profilen

4.4.2.3 Parameter

Parameter sollen es der aufrufenden Applikation ermöglichen, dynamische Daten übergeben zu können. Mit dynamischen Daten sind Daten gemeint, die bei jedem Export / Import ändern und deshalb nicht sinnvoll in einer Profil-Konfiguration gespeichert werden können.
Beispiele dafür sind:

- Aktuelle Patienten-Nummer
- Aktueller Leistungserbringer
- Aktuelles Programm-Datum
- …

Parameter können sowohl während der Definitionszeit (innerhalb der Profil-Konfiguration), als auch während der Laufzeit definiert / hinzugefügt werden (z.B. durch PlugIns). Der Unterschied dieser beiden Arten liegt primär im Verwendungszweck. Parameter, die während der Definitions-Zeit deklariert werden, sollen primär durch die aufrufende Applikation gesetzt und durch die Businesslogik (BL, z.B. Export oder PlugIn) gelesen werden. Im zweiten Fall dienen die Parameter primär als Kommunikation innerhalb der BL (PlugIns) und haben deshalb mit der aufrufenden Applikation direkt „nichts“ zu tun.
Zudem ist es bei den Parametern, die zur Definitions-Zeit deklariert werden, möglich, Default-Werte zu setzen (siehe *IParameterDeclaration*, *IParameter*), die beim Laden einer neuen Profil-Instanz zur Laufzeit automatisch initialisiert werden. Diese Parameter verfügen über folgende Merkmale:

- Name
- Beschreibung (nur in Interface, aber nicht in Konfigurationsfile enthalten)
- Default-Wert

Beispiel:

```
<General>
    <Key>Parameter.CertificateEan</Key>
    <Value>7601001302136</Value>
</General>
```

Code 24: Deklaration von Parametern in Profilen

4.4.2.4 PlugIn Konfiguration

PlugIns benötigen eigene Konfigurationen, deren Struktur das SMEEX-Framework nicht kennt und deshalb auch nicht definieren kann.
Details siehe 4.5.5.

4.4.2.5 Herstellerspezifische Konfiguration

Sofern Herstellerspezifische Konfigurationen notwendig sind (z.B. TSQL- oder Stored Procedure-Konfigurationen), können diese als PlugIn-Konfiguration eingebunden werden.

Beispiel:

```
<Vitodata>
    <Key>senderean</Key>
    <Value />
</Vitodata>
<Vitodata>
    <Key>receiverean</Key>
    <Value />
</Vitodata>
```

Code 25: Deklaration von Hersteller-spezifischen Konfigurationen in Profilen

4.4.3 Profil-Austausch

Profile sollen zwischen den Systemen ausgetauscht werden können. Vor allem Fachverbände sollen Profile selbst definieren können. Angedacht ist, dass die Verbände dann diese Profile Mitgliedern für einen standardisierten Datenaustausch zur Verfügung stellen. Ebenso soll es den Softwareherstellern möglich sein, die durch die Fachverbände definierten Profile um die Hersteller-spezifischen Angaben zu erweitern.
Die Details des Profilaustauschs befinden sich zurzeit noch in Ausarbeitung.

4.4.4 Versionierung

Der Austausch von Profilen birgt auch das Risiko, dass mehrere unterschiedliche Profile mit derselben smeexID im Umlauf sind. **Es gilt daher der Grundsatz, dass einmal definierte Profile nicht verändert werden dürfen!**

Hinweis:
Eine Versionierung ist aus dem Grund nicht vorgesehen, da eine neue Version als neues Profil (neue SmeexID) bereitgestellt wird!

Das Profil beinhaltet trotzdem eine Versionsnummer. Diese soll die Migration zwischen unterschiedlichen SMEEX-Framework-Versionen sicherstellen und kennzeichnet lediglich das technische Format des Profils, und nicht deren Inhalt!

4.4.5 Beispiel-Profil

```
<ProfileConfiguration>
    <General>
        <Key>Name</Key>
        <Value>SMEEX Profile</Value>
    </General>
    <General>
        <Key>DataStructure</Key>
        <Value>&lt;?xml version="1.0" standalone="yes"?&gt;
&lt;smeex xmlns="http://www.smeex.ch/xmlns/smeex/1"&gt;
  &lt;general&gt;
    &lt;name&gt;author&lt;/name&gt;
    &lt;value&gt;UnitTest&lt;/value&gt;
  &lt;/general&gt;
  &lt;general&gt;
    &lt;name&gt;date&lt;/name&gt;
    &lt;value&gt;2009-08-18T20:05:07.6654042+02:00&lt;/value&gt;
  &lt;/general&gt;
  &lt;general&gt;
    &lt;name&gt;generatorId&lt;/name&gt;
    &lt;value&gt;1.2&lt;/value&gt;
  &lt;/general&gt;
  &lt;general&gt;
    &lt;name&gt;id&lt;/name&gt;
    &lt;value&gt;1.2.3&lt;/value&gt;
  &lt;/general&gt;
  &lt;general&gt;
    &lt;name&gt;language&lt;/name&gt;
    &lt;value&gt;de&lt;/value&gt;
  &lt;/general&gt;
  &lt;entries&gt;
    &lt;name&gt;index.xml&lt;/name&gt;
```

```
            &lt;friendlyName&gt;index.xml&lt;/friendlyName&gt;
            &lt;type&gt;index&lt;/type&gt;
            &lt;size&gt;0&lt;/size&gt;
          &lt;/entries&gt;
          &lt;tables&gt;
            &lt;name&gt;T1&lt;/name&gt;
            &lt;friendlyName&gt;Profil-Konfigurationen&lt;/friendlyName&gt;
            &lt;rowCount&gt;0&lt;/rowCount&gt;
            &lt;smeexId&gt;2.16.756.5.30.1.106.1.1&lt;/smeexId&gt;
          &lt;/tables&gt;
          &lt;columns&gt;
            &lt;table&gt;T1&lt;/table&gt;
            &lt;name&gt;C1&lt;/name&gt;
            &lt;friendlyName&gt;Name&lt;/friendlyName&gt;
            &lt;type&gt;string&lt;/type&gt;
            &lt;length&gt;64&lt;/length&gt;
            &lt;isPrimaryKey&gt;true&lt;/isPrimaryKey&gt;
            &lt;isNullable&gt;false&lt;/isNullable&gt;
            &lt;smeexId&gt;2.16.756.5.30.1.106.1.1.1&lt;/smeexId&gt;
          &lt;/columns&gt;
          &lt;columns&gt;
            &lt;table&gt;T1&lt;/table&gt;
            &lt;name&gt;C2&lt;/name&gt;
            &lt;friendlyName&gt;Daten&lt;/friendlyName&gt;
            &lt;type&gt;string&lt;/type&gt;
            &lt;length&gt;65536&lt;/length&gt;
            &lt;isPrimaryKey&gt;false&lt;/isPrimaryKey&gt;
            &lt;isNullable&gt;false&lt;/isNullable&gt;
            &lt;smeexId&gt;2.16.756.5.30.1.106.1.1.2&lt;/smeexId&gt;
          &lt;/columns&gt;
        &lt;/smeex&gt;</Value>
        </General>
        <General>
            <Key>Communicators</Key>
            <Value>Vsfm.Smeex.Framework.PlugIn.FileCommunicator;</Value>
        </General>
        <General>
            <Key>Description</Key>
            <Value>Exportiert alle SMEEX-Profile.</Value>
        </General>
        <Vsfm.Smeex.Framework.PlugIn.FileCommunicator>
            <Key>FileName</Key>
            <Value>SMEEX Profile.smeex</Value>
        </Vsfm.Smeex.Framework.PlugIn.FileCommunicator>
        <Vsfm.Smeex.Framework.PlugIn.FileCommunicator>
            <Key>Path</Key>
            <Value>%USERPROFILE%\Desktop</Value>
        </Vsfm.Smeex.Framework.PlugIn.FileCommunicator>
    </ProfileConfiguration>
```

Code 26: Profil Konfiguration

4.5 PlugIns

Mit PlugIns ist das Verhalten des SMEEX Frameworks steuerbar. Zurzeit können die folgenden PlugIn Arten genutzt werden:

- Kommunikatoren für den Versand an ihren Zielort bzw. die Abholung der Daten von ihrem Quellort
- Transformatoren für die Konvertierung von SMEEX Archiven in andere Datenformate und umgekehrt
- Import-Handler, falls für den Import der Daten ein spezielles Verhalten abweichend vom Standard nötig ist (z.B. für grosse Datenmengen)
- Export-Handler, falls für den Export der Daten ein spezielles Verhalten abweichend vom Standard nötig ist (z.B. für grosse Datenmengen)

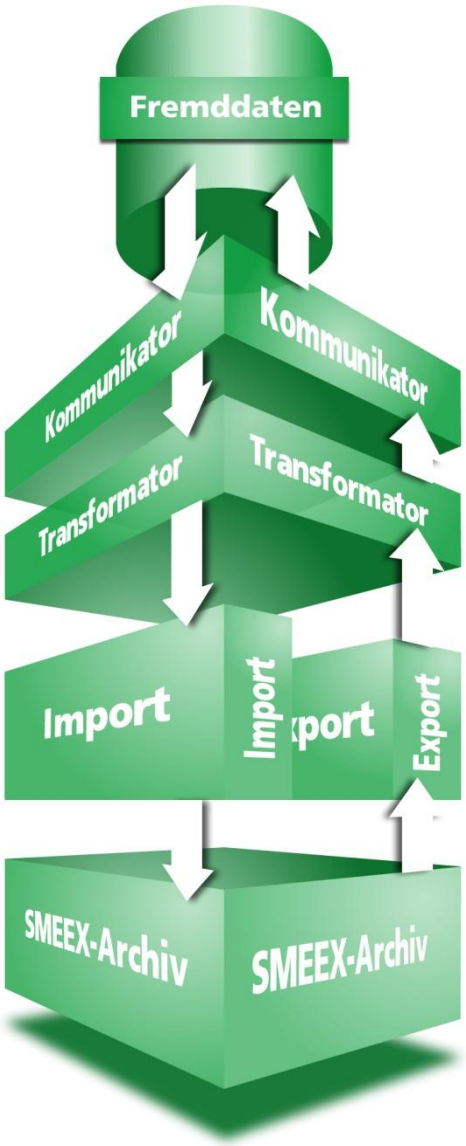

Abbildung 12: Verarbeitungsreihenfolge der PlugIns

4.5.1 Transformatoren

Mittels dieses PlugIns können SMEEX-Daten in gebräuchliche Datenformate transformiert werden. Unter gebräuchlich werden derzeit die Formate, clinical document architecture *(CDA)*, cross enterprise document sharing *(XDS)* und portable document format *(PDF)* verstanden. Es ist davon auszugehen, dass im Laufe der Zeit weitere verschiedene Formate implementiert und unterstützt werden.

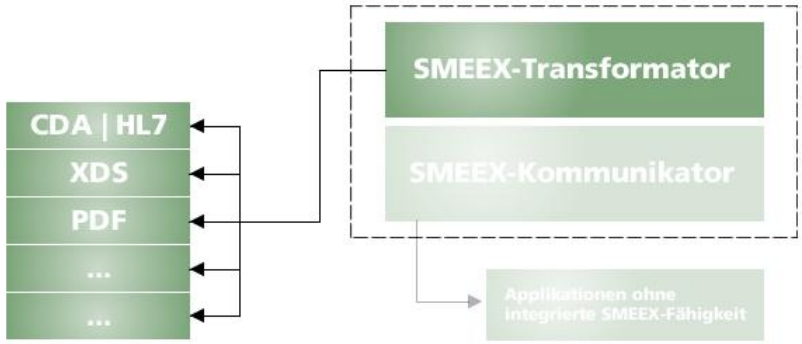

Abbildung 13: Transformator PlugIn

Hinweis:
In der Version 1.0 des SMEEX Frameworks sind noch keine Transformator-PlugIns enthalten. Da Transformatoren ein integraler Bestandteil der Architektur des SMEEX Frameworks sind, werden sie der Vollständigkeit halber bereits hier beschrieben.

4.5.2 Kommunikatoren

Dieses PlugIn stellt die elektronische Datenkommunikation sicher und stellt sämtliche Funktionalitäten die für den gesicherten Datenaustausch notwendig sind zur Verfügung. Verschiedene Anbieter können einen Kommunikationsdienst entwickeln und selbst im Markt anbieten.

Hinweis:
Für den Verkehr zwischen SMEEX-Applikationen ist grundsätzlich kein Kommunikator erforderlich (ausser beim Bedarf, einen verschlüsselten, sicheren Kommunikationsweg einzuhalten). Ein Austausch kann beispielsweise auch über ein Speichermedium (Memory-Stick etc.) erfolgen.

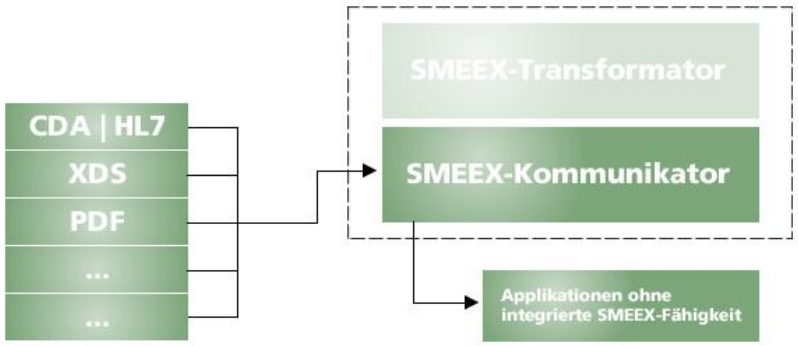

Abbildung 14: Kommunikator PlugIn

4.5.3 Import-Handler

Dieses PlugIn ersetzt den Standard Import-Handler. Es kommt zur Anwendung, wenn für den Import ein spezielles Vorgehen nötig ist, zum Beispiel ein Handling bei grosser Datenmengen.

4.5.4 Export-Handler

Dieses PlugIn ersetzt den Standard Export-Handler. Es kommt zur Anwendung, wenn für den Export ein spezielles Vorgehen nötig ist, zum Beispiel ein Handling bei grossen Datenmengen.

4.5.5 PlugIn Konfiguration

Aus Benutzersicht folgt vielfach bei einem Export auch der Versand des exportierten Files. PlugIns benötigen meist eigene Konfigurationen, deren Struktur das SMEEX-Framework nicht kennt und deshalb auch nicht definieren kann. Es soll daher für die PlugIn-Mechanismen eine Möglichkeit bestehen, zusätzliche Konfigurationen (z.B. für einen Fileversand) im Profil zu hinterlegen (siehe auch Kapitel 0).

Dafür wird innerhalb des Profils ein Mechanismus bereitgestellt, mit dessen Hilfe beliebige Key-Value-Pairs gespeichert werden können (getrennt nach Sektionen). Es ist daher jedem einzelnen PlugIn überlassen, welche Konfiguration wie gespeichert werden sollen.
Die Sektion dient dabei als „schwache“ Trennung und Gruppierung der Einträge. PlugIn-Hersteller sollten einen Sektions-Namen verwenden, der möglichst eindeutig ist (z.B. bestehend aus Hersteller- und Produkt-Name). Werden Einträge gemeinsam verwendet, so kann auch eine gemeinsame Sektion verwendet werden.

Hinweis:
Sollen gemeinsame Sektionen von PlugIns unterschiedlicher Hersteller verwendet werden, so ist eine Absprache unter diesen Herstellern zu empfehlen.

Achtung:
Falls sicherheitsrelevante Informationen durch PlugIns gespeichert werden sollen, müssen diese durch das PlugIn selbst geschützt werden (Verschlüsselung etc.)!

Beispiel:

```
<Vsfm.Smeex.Framework.PlugIn.FileCommunicator>
    <Key>FileName</Key>
    <Value>PDF-Export.smeex</Value>
</Vsfm.Smeex.Framework.PlugIn.FileCommunicator>
<Vsfm.Smeex.Framework.PlugIn.FileCommunicator>
    <Key>Path</Key>
    <Value>%USERPROFILE%\Desktop</Value>
</Vsfm.Smeex.Framework.PlugIn.FileCommunicator>
```

Code 27: Beispiel für die PlugIn-Konfiguration in Profilen

4.6 smeexIDs

4.6.1 Motivation

Um Daten verlässlich über Systemgrenzen hinweg austauschen zu können, muss jedes Datenfeld eine eindeutige Identifikation besitzen. Diese Identifikation muss verbindlich und unveränderbar sein (d.h. wenn einem Feld, das den Nachnamen beinhaltet, einmal eine bestimmte ID zugewiesen wurde, muss ein Feld mit dieser ID immer und ausschliesslich den Nachnamen enthalten).
Für den Austausch medizinischer Daten benötigt man also eine ID-Systematik, die allen relevanten Datenfeldern eine solche eindeutige ID zu weisst. Da es bisher keine solche ID-Systematik gibt, die alle nötigen medizinischen und administrativen Daten umfasst, hat der VSFM die smeexIDs spezifiziert.

Um die Arbeit mit den smeexIDs zu erleichtern, wird der smeexIDManager (siehe 4.6.3) zur Verfügung gestellt. Er bietet einen komfortablen Zugriff auf das smeexID Repository.

4.6.2 Spezifikation der smeexIDs

Die smeexID ist eine Objekt ID, die jedes Datenfeld innerhalb des SMEEX Datenformats eindeutig identifiziert. Sie entspricht einer erweiterten OID-Definition, wobei der smeexID neben der „OID“ Nummer weitere Attribute wie ein Datentyp, ein optionales Pattern und eine Kardinalität zugewiesen werden. Die Root ID der smeexIDs ist 2.16.756.5.30.1.106.1.10.
Der SMEEX Standard wird vom VSFM gepflegt. Das beinhaltet die Vergabe der smeexIDs und die Aufnahme neuer Datenfelder in den SMEEX Standard.
Für Datenfelder, die nicht im SMEEX Standard enthalten sind, können herstellerspezifische IDs vergeben werden. Die Pflege und Verwendung dieser herstellerspezifischen IDs erfolgt nicht über den SMEEX ID Manger, sie müssen individuell umgesetzt werden.

Direkt unter dem VSFM Knoten (@root = 2.16.756.5.30.1.106) können im Knoten „members“ (ID = @root.3) interessierten Herstellern ein eigener Knoten zugewiesen werden. Die Vergabe der Hersteller-IDs erfolgt durch VSFM. Unterhalb des eigenen Knotens kann ein Hersteller frei IDs vergeben.

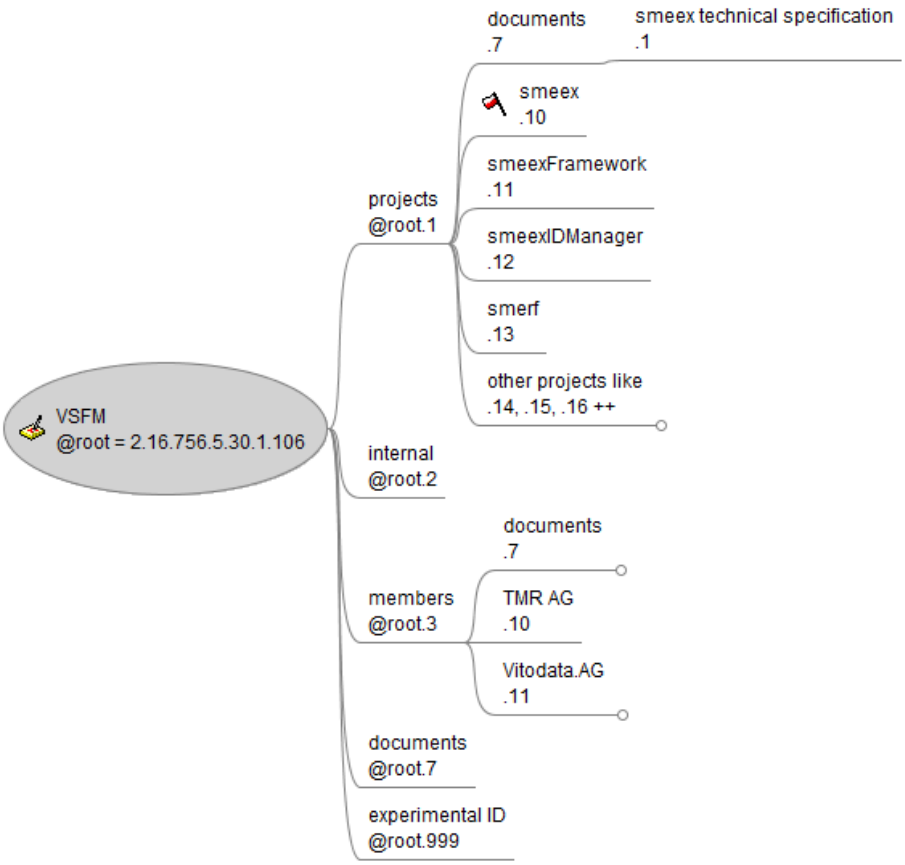

Abbildung 15: Aufbau der ID Struktur unterhalb des offiziellen VSFM Knotens

4.6.3 smeexIDManager

Der smeexIDManager ist ein Tool zur Verwaltung der smeexIDs. Er dient als Nachschlagewerk für den SMEEX Standard. Für jedes Datenfeld des Standards sind smeexID, Name, Version, Datentyp, Pattern und Kardinalität sowie eine Beschreibung des Inhalts hinterlegt (siehe Abbildung 16).
Die Daten des SMEEX Standards, auf die der smeexIDManager zugreift, sind in einer MS Access 2007 Datenbank abgelegt, die mit dem Manager mitgeliefert wird.

Im vorliegenden Dokument wird nur ein kurzer Überblick über den smeexIDManager gegeben, weitergehende Informationen sind in der Dokumentation des smeexIDManagers zu finden.
Im smeexIDManager wird im Tab-Register „Definition“, die komplette Struktur des ausgewählten SMEEX-Standard Profils dargestellt. Im smeexIDManager sind verschiedene vordefinierte smeexProfile (siehe 4.4.1) für die gängigsten Anforderungen enthalten. Dies umfasst zum Beispiel das Profil „fullSMEEX“, das alle im SMEEX Standard definierten IDs enthält oder auch Profile, die zum Beispiel die Daten eines einzelnen Patienten für die Kommunikation umfassen.

Zu jedem Datenfeld sind alle hinterlegten Informationen (ID, Name, Version, Datentyp, Pattern, Kardinalität und Beschreibung) abrufbar. Wählt man im Baum auf der linken Seite das entsprechende Datenfeld, werden die Informationen dazu auf der rechten Seite angezeigt.

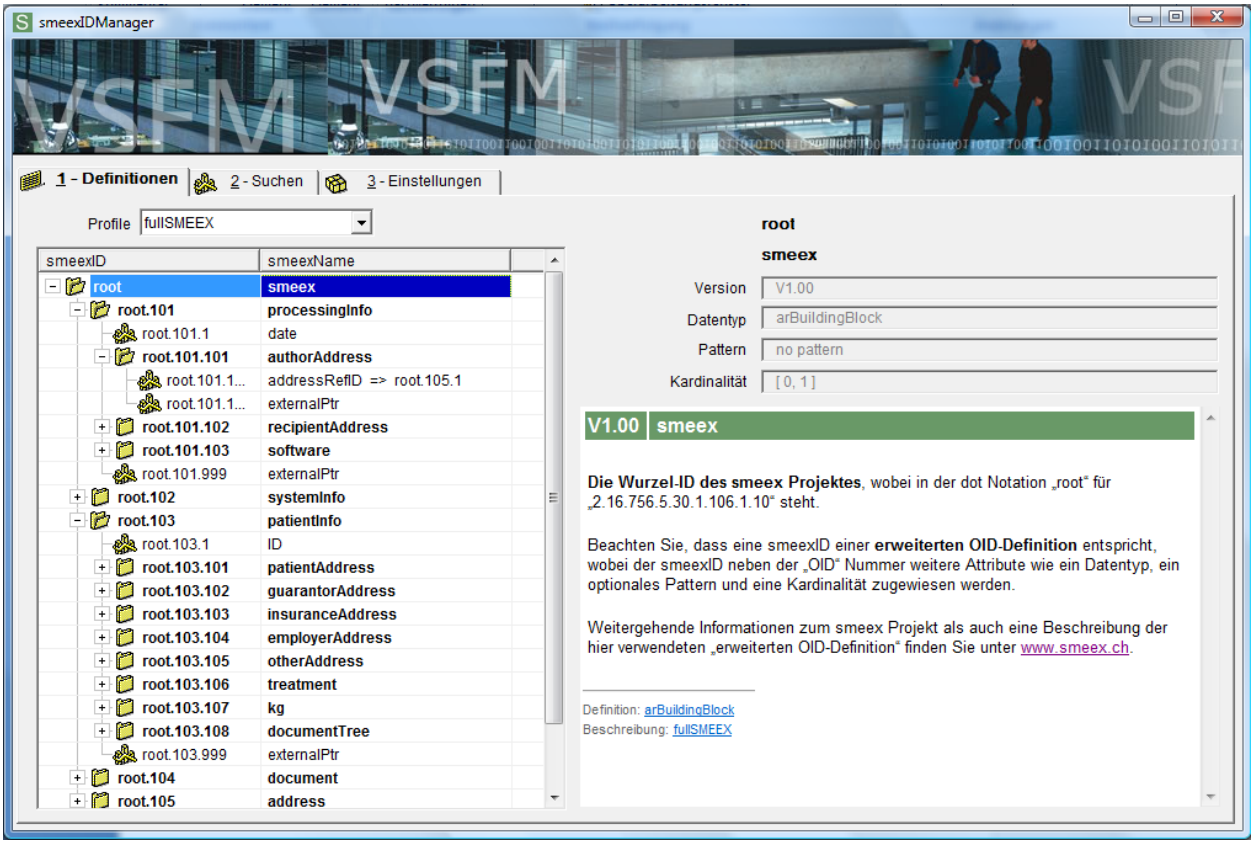

Abbildung 16: smeexIDManager, Tab-Register Definition

Im Tab-Register „Suchen“ kann man gezielt nach einzelnen Datenfeldern suchen. Suchkriterien sind Änderungsdatum, smeexID (z.B. root.101) oder eine Zeichenfolge, die im Namen des Datenfelds oder optional auch in der Beschreibung des Datenfelds vorkommt.

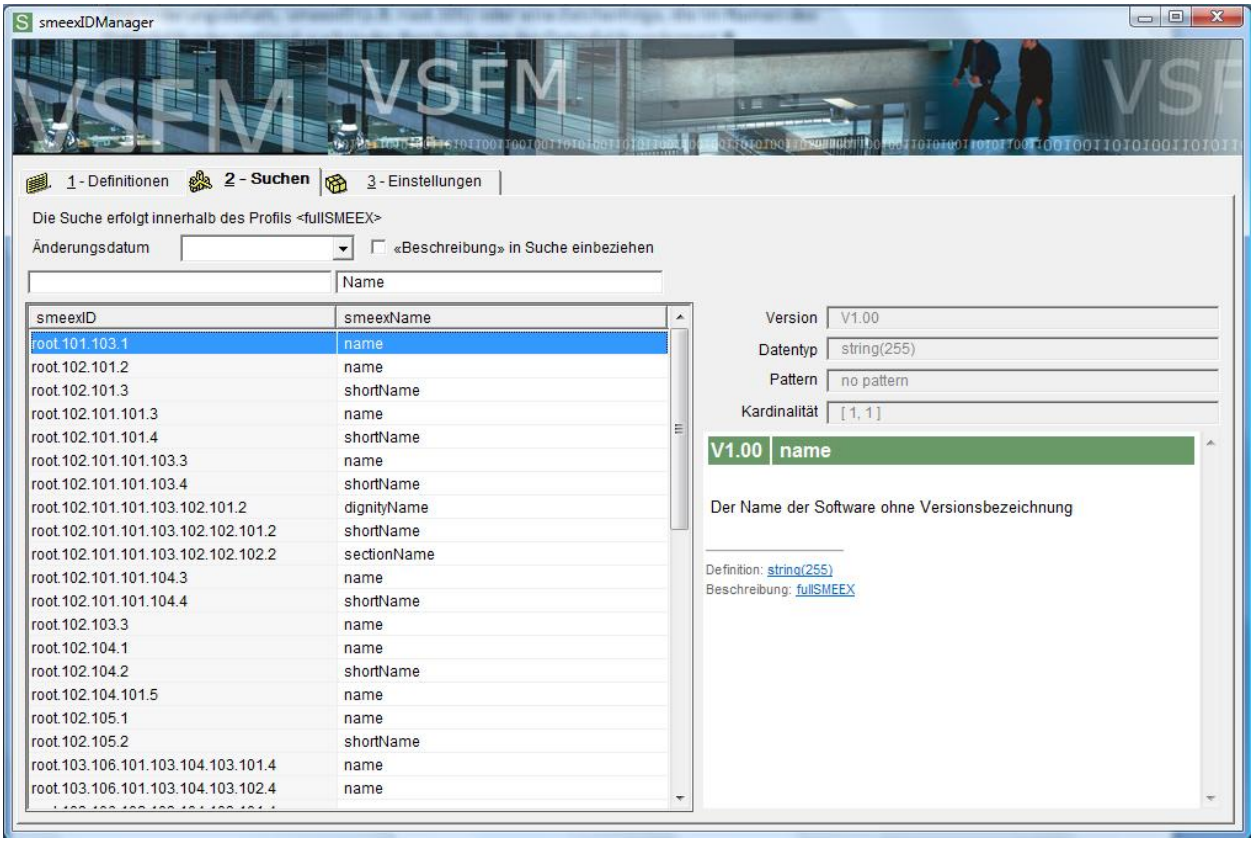

Abbildung 17: smeexIDManager, Tab-Register Suchen

Die Informationen zu den einzelnen Knoten können als HTML Dokument exportiert werden. Dazu klickt man im Tab-Register „Definition“ mit der rechten Maustaste auf den gewünschten Knoten und wählt „Export Knoten“ aus. Darauf hin öffnet sich ein Browserfenster, in dem das exportierte HTML-File angezeigt wird. Sie enthält die alle Datenfeldinformationen zu dem ausgewählten Knoten und allen seinen Kindern.

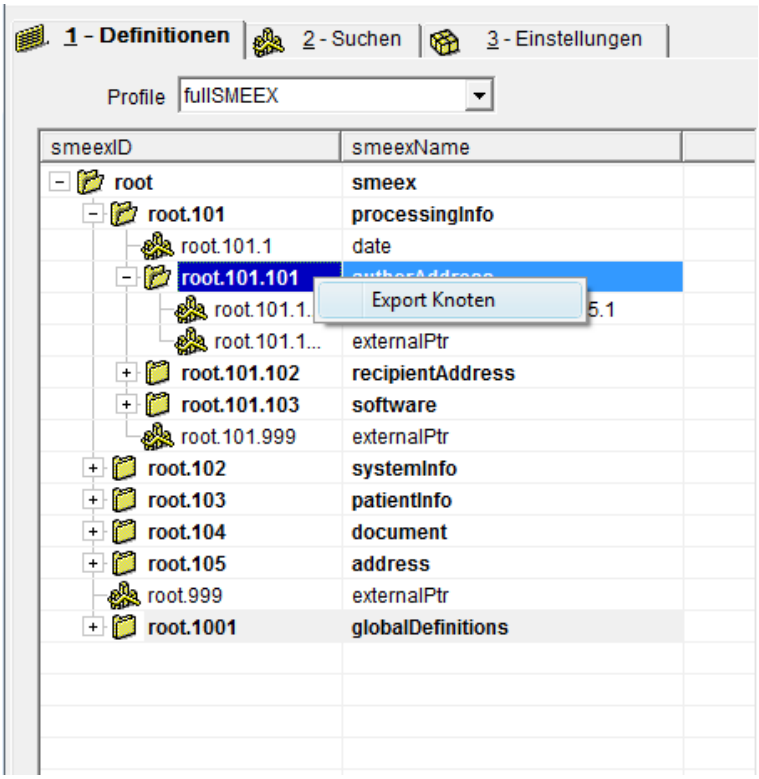

Abbildung 18: smeexIDManager, Knoten exportieren

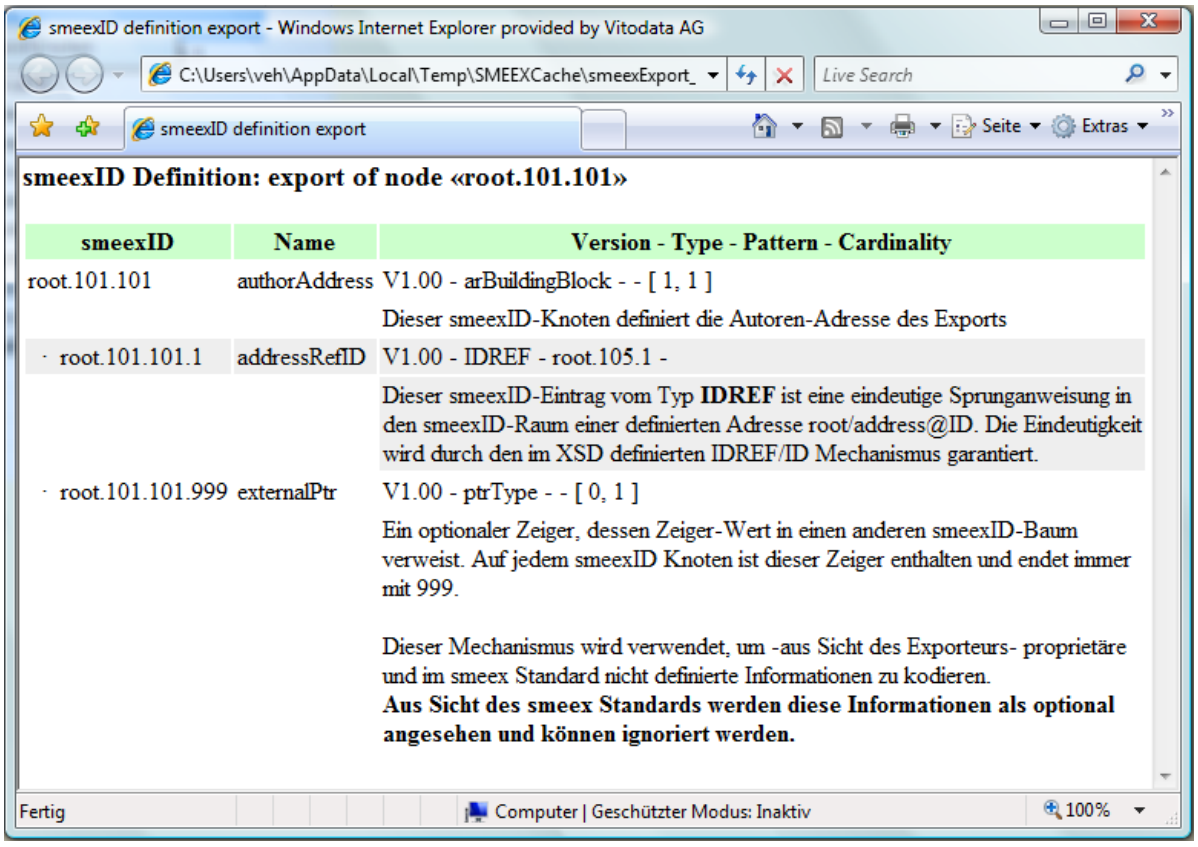

Abbildung 19: smeexIDManager, Resultat Knoten Export

4.7 Swiss MEdical ReFerence system (SMERF)

4.7.1 Motivation

Um Daten über die Systemgrenzen hinweg u.v.a. automatisiert austauschen zu können, sind nebst eindeutigen Referenzsystemen auch Klassifizierungs- und Codierungsysteme für Labor- und klinische Messresultate notwendig. Da bzgl. der elektronischen Patientenaktenführung keine allgemeingültigen Standards für die Dokumentation von medizinischen und klinischen Daten zur Verfügung stehen, bestehen umfangreiche technische Herausforderungen, um den Datenaustausch für den Endanwender nutzbringend gewährleisten zu können.
Beim Datenaustausch zwischen einem Sende- und Empfängersystem, erzeugen die Daten auf Empfangsseite nur dann einen Mehrwert, wenn eine lückenlose Interpretation erfolgen kann. Genau die angesprochene Interpretation bildet dabei die Problemstellung. Um die Problemstellung zu verdeutlichen wird folgendes Beispiel gewählt:

Das Sendesystem sendet ein klinisches Messresultat mit dem Inhalt: '120 / 80'. Es stellt sich nun die Frage, wie das Empfangssystem damit umgehen soll? Das gelieferte Resultat ist in seiner Klassifikation und Spezifikation unzureichend definiert, als dass es automatisiert durch ein IT-System verarbeitet werden könnte. Damit eine minimale Verarbeitung erfolgen kann, muss zusätzlich eine Aussage über die Bedeutung und zur Masseinheit gemacht werden. Je nach Messresultat sind weitere Attribute und Spezifikationen notwendig, um eine abschliessende Interpretation vorzunehmen.

Erst wenn technische Möglichkeiten geschaffen werden, Laborwerte und klinische Messresultate voll zu qualifizieren, dann entstehen für den Endanwender nutzbare Informationen.

Das SMERF Projekt nimmt sich den vorgängig beschriebenen Problemstellungen im Bezug auf die Interpretation von Labor- und klinischen Messresultaten an. Dabei wird das Ziel verfolgt, ein einfaches, erweiterbares und im medizinischen Alltag handhabbares System für die Klassifizierung- und Codierung zur Interpretation zu schaffen. Vorhandene Systeme (wie LOINC, SNOMED, u.a.) werden in der Konzeption berücksichtigt. Es wird aber besonderen Wert auf eine zweckmässige Integration in medizinische Branchenapplikationen und die Handhabbarkeit im Endanwendungsalltag gelegt.

Es ist zu beachten, das SMERF keine Konkurrenz zu z.B. Snomed CT darstellt. Solche Kataloge qualifizieren und repräsentieren die inhaltlichen Elemente eines klinischen Resultats. Die analytische Präzision und Mächtigkeit eines Katalogs wie Snomed CT (18 Achsen, 800'000 Terms, 300'000 Konzepte!) hat natürlich seinen Preis in finanzieller, zeitlicher und intellektueller Hinsicht. Daher wird mindestens im Moment SMERF als Codierungssystem in SMEEX verwendet.

Zielsetzung
SMERF ist ein Klassifizierungssystem für:

- Laborwerte
- klinische Messresultate

Minimal werden die Bedeutung und die Masseinheit (UCUM-Einheiten) klassifiziert. Je nach Bedeutung des Resultats, stehen weitere Klassifikationsattribute zur Verfügung. Das System ist vertikal beliebig erweiterbar. Wie im SMEEX Standard werden die einzelnen Attribute eindeutig über eine ID, genannt smerfID, identifiziert.

Grundsätzlich ist es dem Sendesystem überlassen, wie detailliert die Daten klassifiziert werden. Je granularer die Klassifizierung durchgeführt wird, desto „wertvoller" sind die Daten auf Empfangsseite.

Beispiel:
Um die Zielsetzung zu veranschaulichen, wird obiges Beispiel nochmals aufgegriffen.

Fall 1:
Versand des Datenfelds mit dem Inhalt „120 / 80" ohne weitere Klassifizierung.

Fall 2:
Versand des Datenfeldes mit dem Inhalt „120", Klassifizierung: Blutdruck, [Pa], systolisch, links, sitzend
Versand des Datenfeldes mit dem Inhalt „80", Klassifizierung: Blutdruck, [Pa], diastolisch, links, sitzend

Bezüglich des Falls 1 kann keine Aussage gemacht werden, um welchen Feldinhalt es sich handelt. Eine Datenweiterverarbeitung ist nicht möglich. In einem solchen Fall kann das Empfangssystem die erhaltenen Daten nur in ein Textdatenfeld ablegen und dem Systembenutzer die weitere Verarbeitung überlassen.

Ganz anders im Fall 2. Hier wird eine volle Klassifizierung des Dateninhaltes vorgenommen. Eine weitere Verarbeitung durch das System ist grundsätzlich möglich.

4.7.2 SMERF Spezifikation

Abbildung 20: Gesamtsicht Aufbau des SMERF

Innerhalb von smerf werden 2 Typen von Knoten definiert:

- **Attribut-Knoten** mit Endknoten-Nummern zwischen 1 und 99, welche nur weitere Attribut-Subknoten enthalten können (Symbol im smerfIDManager:)
- **Objekt-Knoten** mit Endknoten-Nummern > 99, welche Attribut- und Objekt-Knoten enthalten können. (Symbol im smerfIDManager: aufgeklappt bzw. zugeklappt oder ohne Sub-Knoten)

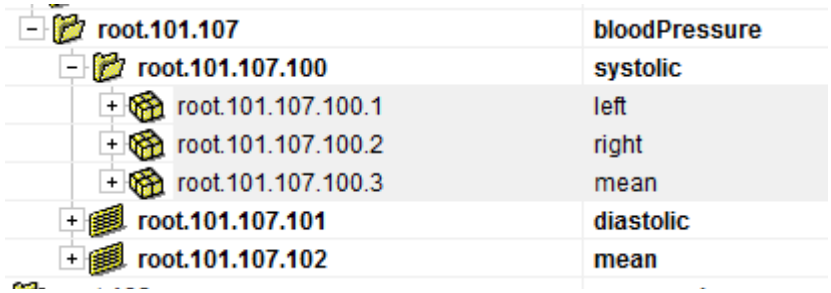

Abbildung 21: Knoten-Struktur in SMERF

Aus inhaltlicher Sicht sind die beiden Typen nicht unterscheidbar, die Art der Erweiterung definiert den Unterschied:

- ein weiterer Objekt-Knoten kann an alle anderen Objekt-Knoten hinzugefügt werden und erweitert dabei den übergeordneten Objekt-Knoten um ein weiteres spezielles Objekt.
- die Erweiterung einer Attribut-Achse (der Menge aller Attribut-Knoten einer spezifischen Ebene) um einen weiteren Attribut-Knoten bedeutet, dass der neue Attribut-Knoten alle Subachsen erbt, welche die anderen Knoten dieser Achse bereits besitzen.

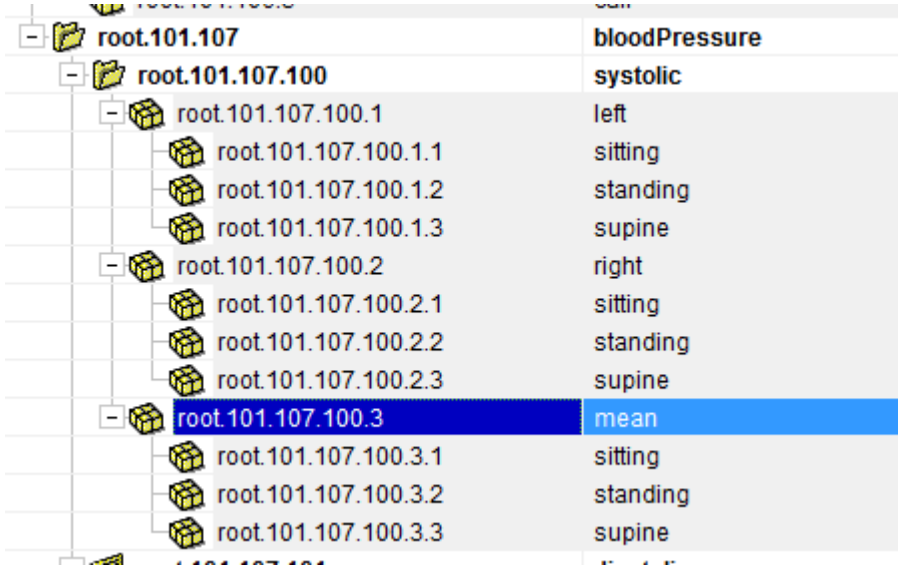

Abbildung 22: Beispiel einer Attribut-Achse, alle Knoten haben die gleichen Subachsen

In Abbildung 22 wird die Attribut-Achse aus den Attribut-Knoten *left*, *right* und *mean* gebildet. Die Subachsen sind *sitting*, *standing* und *supine*. Das in der Abbildung benutzte Beispiel dient nur zur Veranschaulichung der Subachsen. Es ist in der Form nicht im smerfIDManager enthalten.

Jeder Knoten in SMERF ist eindeutig durch die smerfID identifizierbar. Jedem Knoten wird neben einem Namen (smerfName) allenfalls weitere Informationen wie die UCUM Einheit und äquivalente LOINC-Codes zugewiesen.

Beispiel: Grösse

smerfID:	root.101.100
smerfName:	height
UCUM Einheit:	m
Äquivalente LOINC:	z.B. 8305-5, 8306-3, 8307-1, 8308-9

4.7.3 Spezifikation der smerfIDs

Die smerfID ist eine Objekt ID, die jedes Element innerhalb des SMERF eindeutig identifiziert. Sie entspricht einer erweiterten OID-Definition. Die Root ID der smerfIDs ist 2.16.756.5.30.1.106.1.13.
Das SMERF wird vom VSFM gepflegt. Das beinhaltet die Vergabe der smerfIDs und die Aufnahme neuer Knoten in das SMERF.

4.7.4 smerfIDManager

Im vorliegenden Dokument wird nur ein kurzer Überblick über den smerfIDManager gegeben, weitergehende Informationen sind in der Dokumentation des smerfIDManagers zu finden.
Im smerfIDManager wird im Tab-Register „Definition“, die komplette Struktur des SMERF dargestellt.
Zu jedem Knoten werden allenfalls die UCUM Einheit und äquivalente LOINCs angezeigt. Wählt man im Baum auf der linken Seite das entsprechende Datenfeld, werden die Informationen dazu auf der rechten Seite angezeigt.

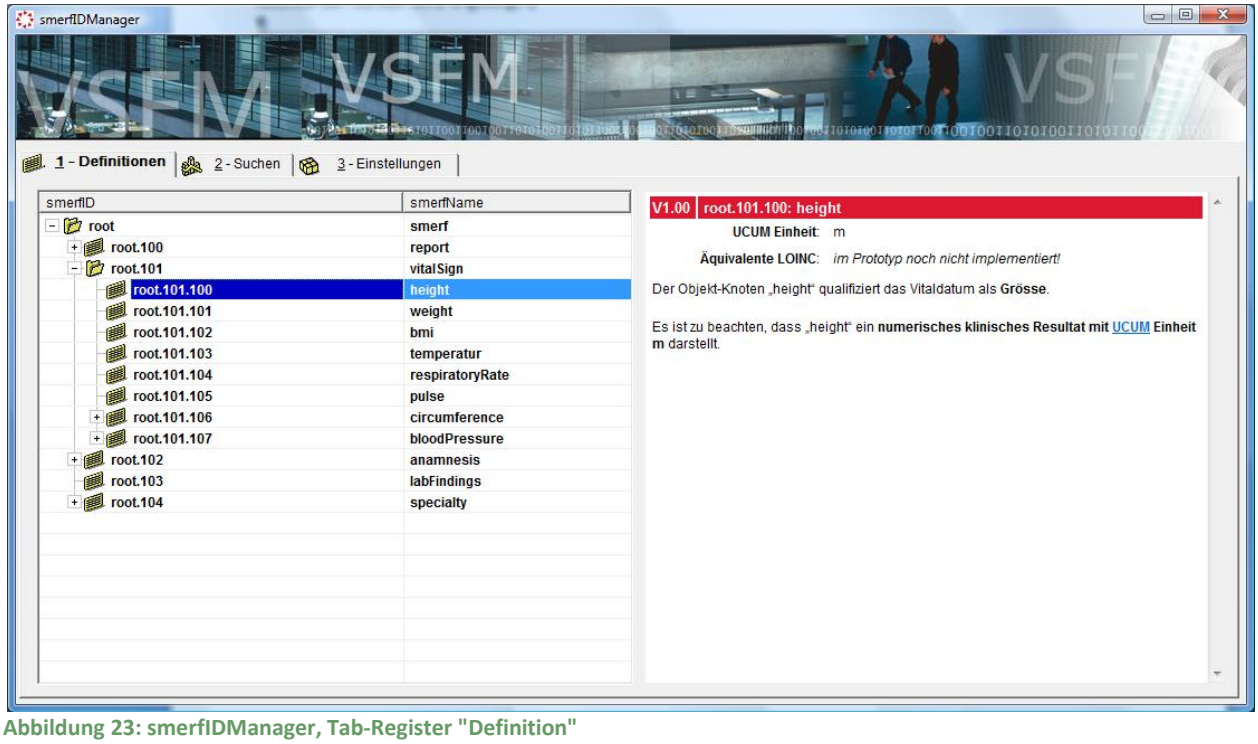

Abbildung 23: smerfIDManager, Tab-Register "Definition"

Im Tab-Register „Suchen“ kann man gezielt nach einzelnen Knoten suchen. Suchkriterien sind Änderungsdatum, smerfID (z.B. root.101) oder eine Zeichenfolge, die im Namen des Knoten oder optional auch in seiner Beschreibung vorkommt.

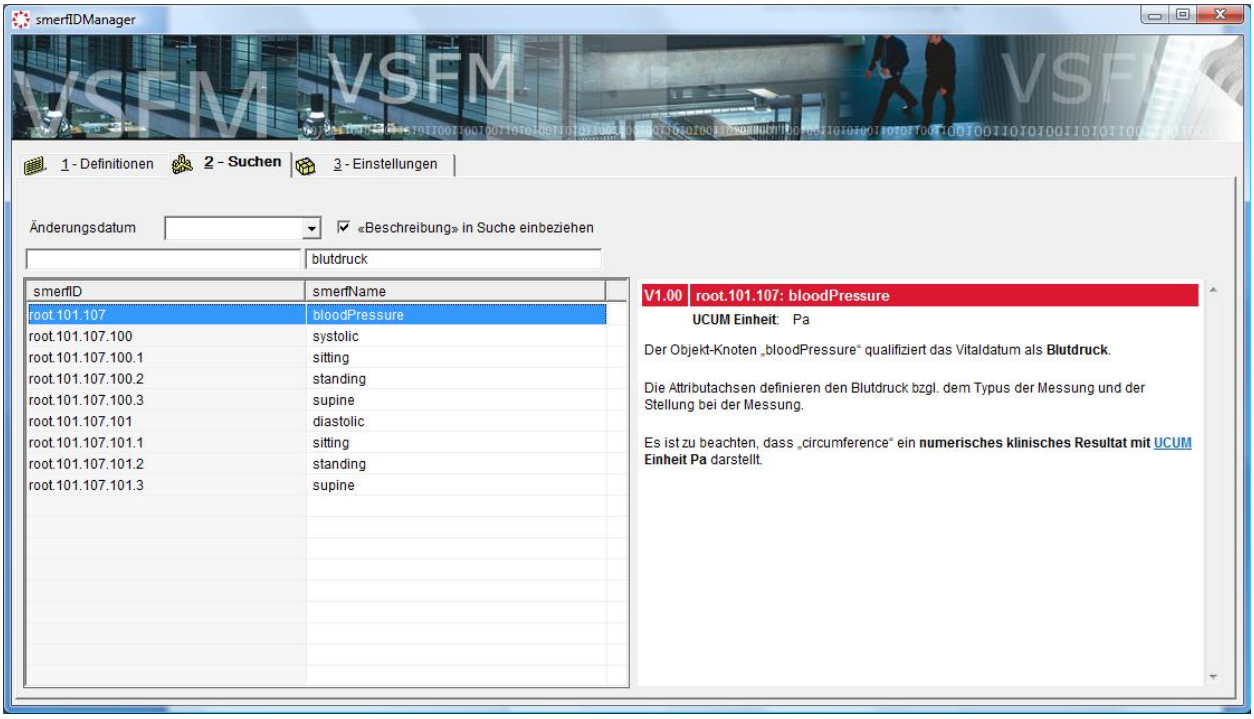

Abbildung 24: smerfIDManager, Tab-Register "Suchen"

4.8 Demo-Applikation

Die SMEEX-Demo-Applikation dient dem Zweck, die SMEEX-Framework Integration für Softwarehäuser zu erleichtern und mit Quellcodebeispielen die Anwendung zu demonstrieren. Die Applikation stellt eine für das Schweizerische Gesundheitswesen gängige Datenbasis dar. Datenstruktur wie auch Dateninhalt sind weder komplett noch umfassend.
Mit der Applikation werden Quellcodebeispiele mitgeliefert, welche uneingeschränkt für Integrationszwecke kostenlos genutzt werden können.

Hinweis:
Die in der Demo-Applikation hinterlegten Daten stellen keinen kompletten Datenbestand dar. Dies gilt sowohl für die Struktur der erfassten Daten, als auch für ihren Inhalt und Umfang. Die Datenbasis der Demo-Applikation ist lediglich als Ausschnitt zu sehen, der die Möglichkeiten des SMEEX Frameworks veranschaulichen soll.
Bitte beachten Sie hierzu auch die Hinweise im „Lies mich" Fenster der Demo-Applikation, das beim Start angezeigt wird.

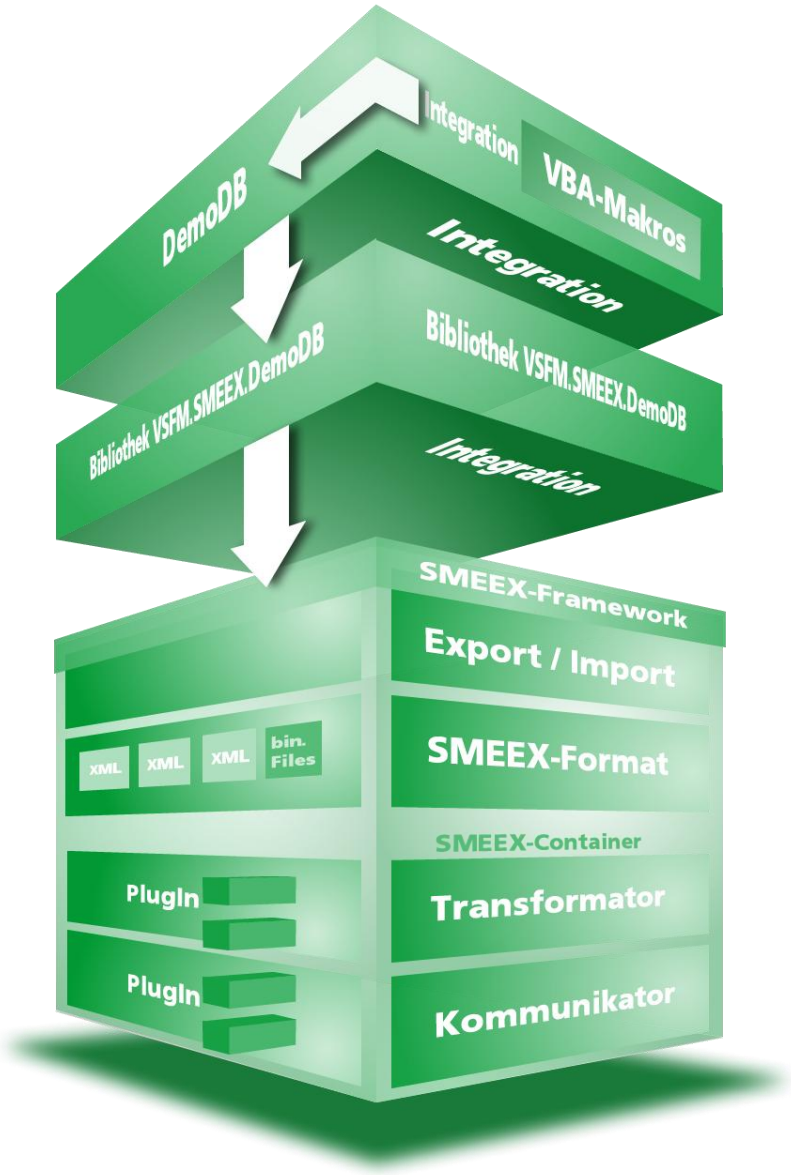

Abbildung 25: Komponenten der Demo-Applikation

Informationen zu den Systemvoraussetzungen für den Betrieb der Demo-Applikation siehe 3.1.

4.8.1 Start der Demo Applikation

Um die Demo-Applikation zu starten, muss lediglich die enthaltene MS Access Datenbank SMEEX.accdb in MS Access 2007 (oder höher) geöffnet werden. Eine Installation ist nicht notwendig.
Abhängig von der eingestellten Sicherheitsstufe, wird der folgende Dialog für die Aktivierung der enthaltenen Makros angezeigt:

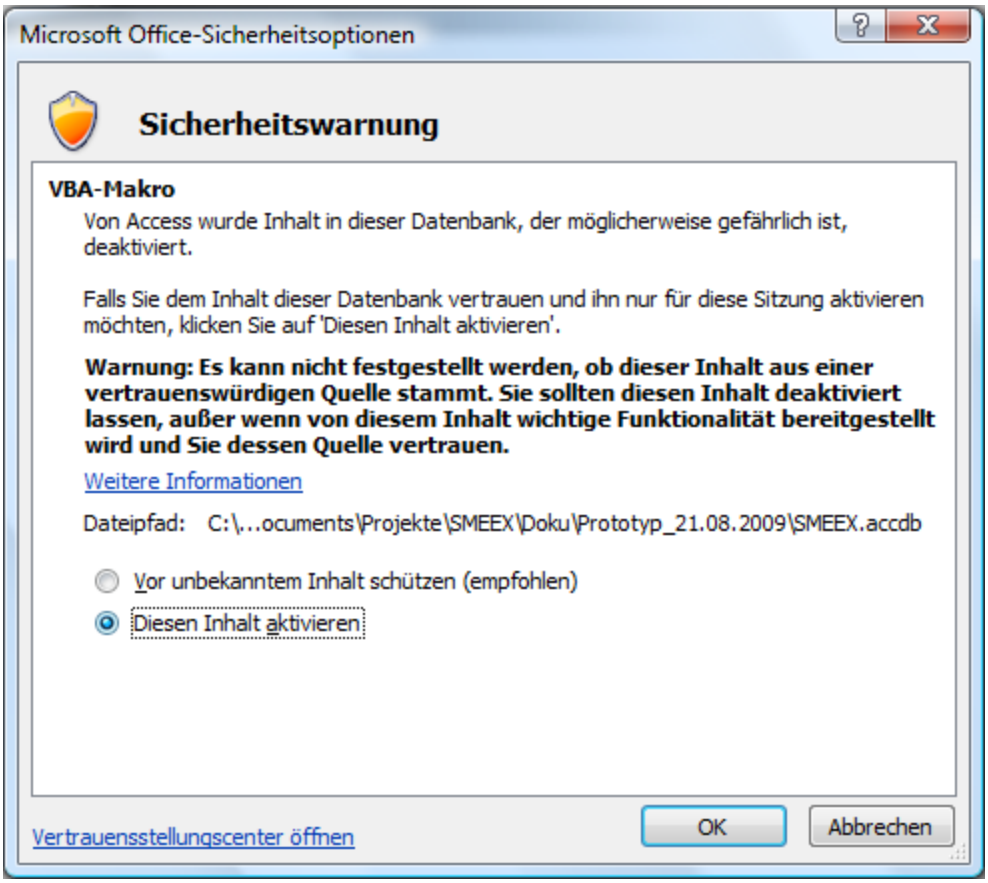

Abbildung 26: Demo-Applikation: Sicherheitswarnung beim Start

Wird der Dialog angezeigt, „Diesen Inhalt aktivieren“ wählen und auf OK klicken.
Es wird ein „Lies mich“-Dialog angezeigt, der mit einem Klick auf „Weiter“ geschlossen werden kann. Dann wird das Hauptfenster der Demo Applikation angezeigt (siehe 4.8.2).

4.8.2 Hauptdialog

Im Hauptdialog ist links eine Baum mit den in der Demo-Applikation hinterlegten Stammdaten. Im rechten Teil des Dialogs befinden sich die Steuerelemente, mit denen die eigentliche Funktionalität des SMEEX Frameworks demonstriert wird.
Als Profil für Ex- und Import kann eines der vordefinierten Profile gewählt werden. Werden in den Feldern „Verzeichnis“ und „Dateiname“ kein Wert angegeben, werden die im Profil hinterlegten Daten für das Kommunikator PlugIn verwendet. Ist kein Kommunikator im Profil hinterlegt, werden die exportierten Daten wieder verworfen. Ein solches Vorgehen eignet sich zur Validierung des Exportprofils, da der Export selbst stattfindet und allfällige Fehler ausgegeben werden.
Ein Klick auf den entsprechenden Button startet den Export resp. den Import.

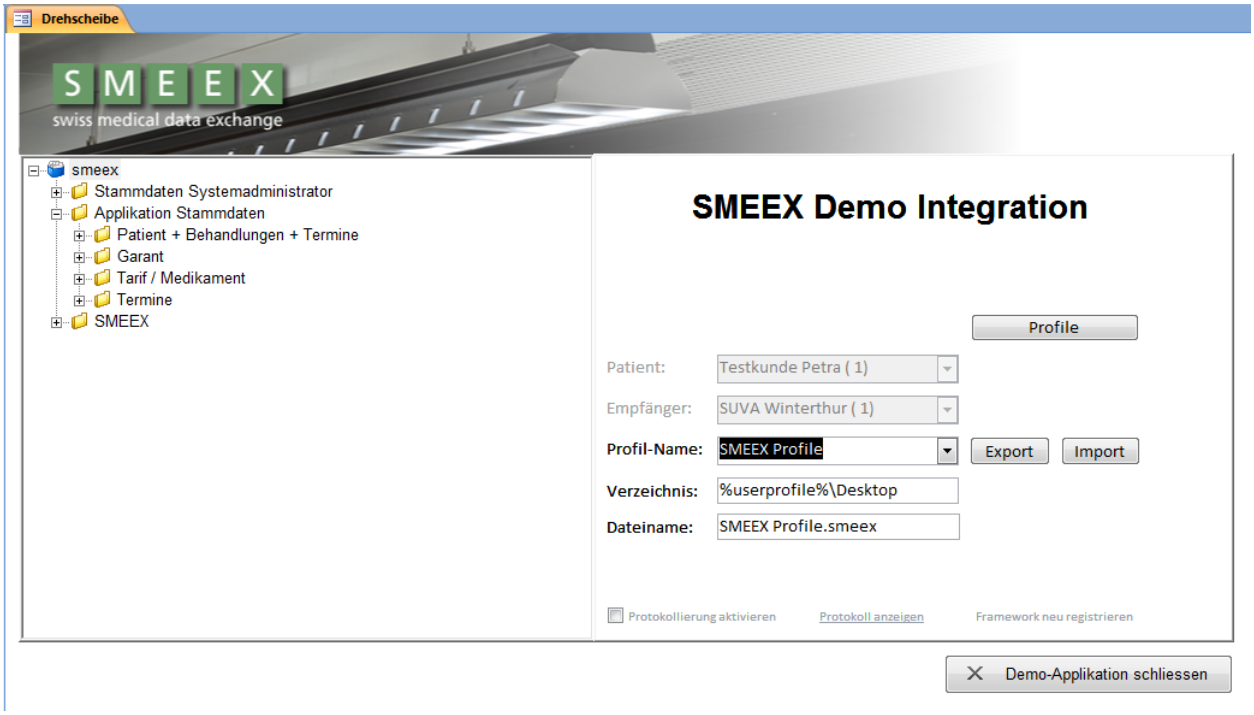

Abbildung 27: Demo-Applikation: Hauptdialog

4.8.3 Stammdatenpflege

Die in der Datenbank hinterlegten Stammdaten können über Pflegedialoge verändert und erweitert werden. Dazu muss man im Hauptdialog im Baum auf der linken Seite die entsprechenden Stammdaten auswählen und doppelt anklicken.

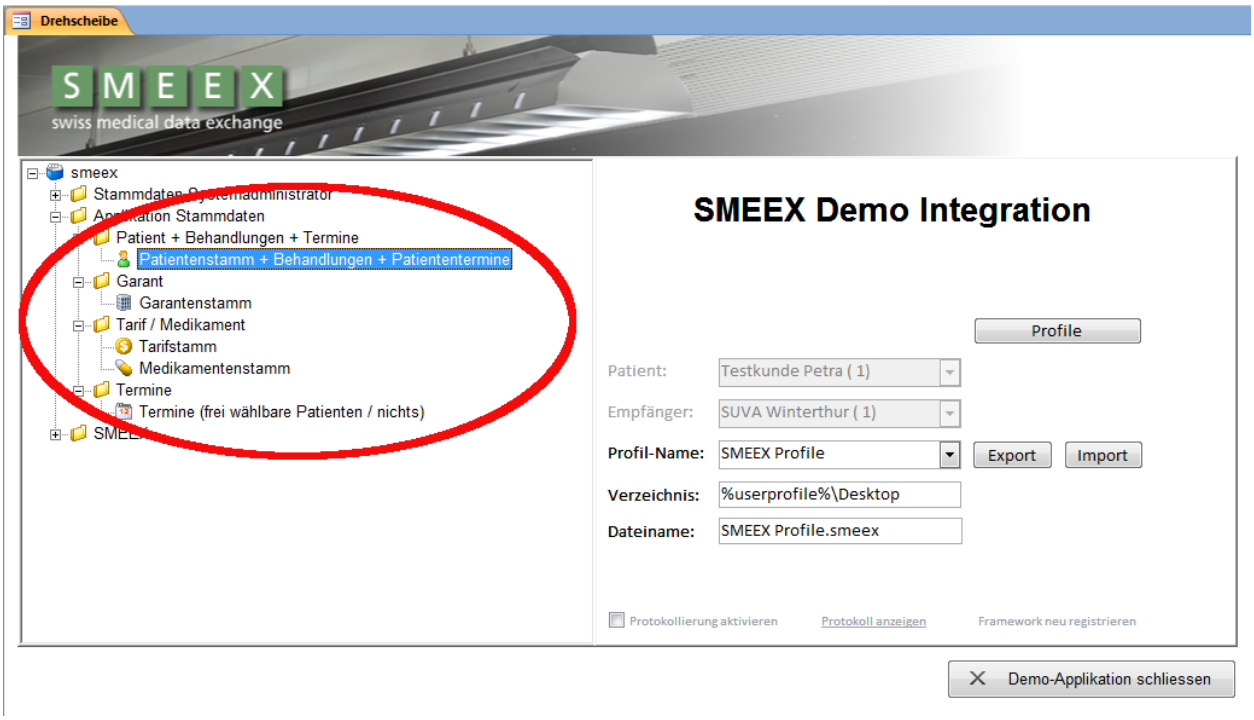

Abbildung 28: Demo-Applikation: Stammdaten-Auswahl

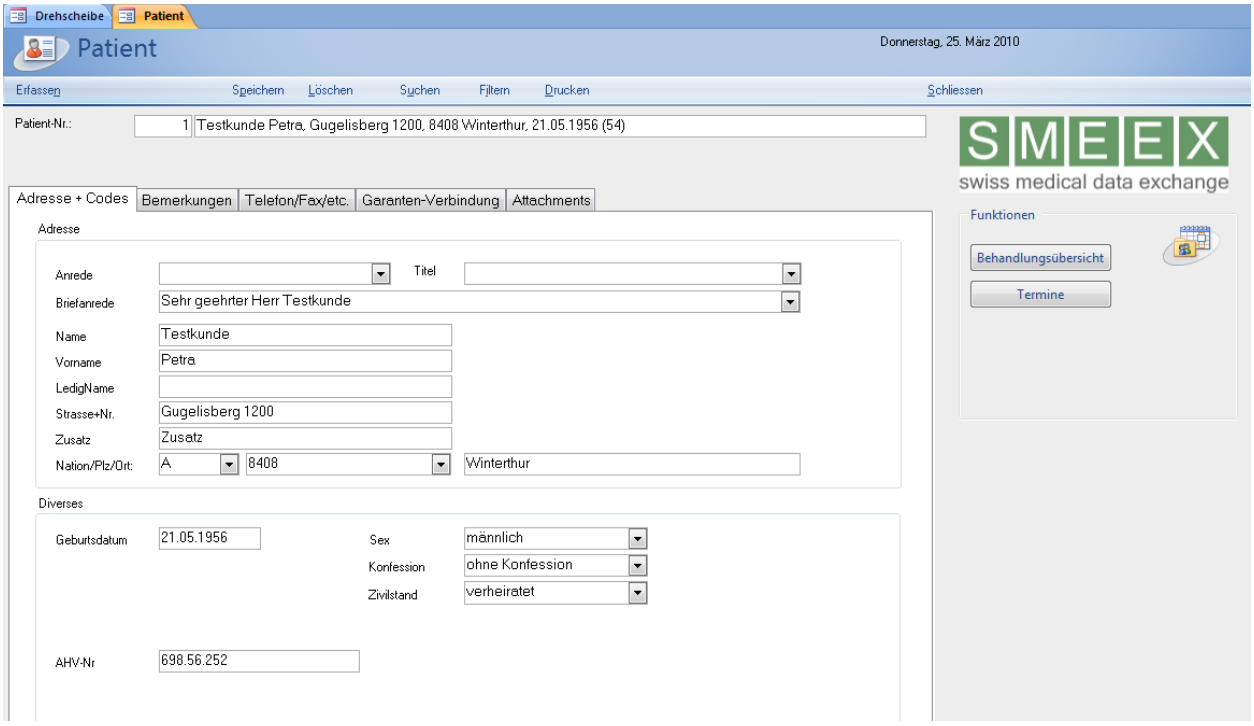

Abbildung 29: Demo-Applikation: Beispiel für Stammdaten-Pflege-Dialog

4.8.4 Profil-Konfiguration

Die für Ex- bzw. Import verwendeten Profile können in einem eigenen Dialog Profil-Konfiguration gepflegt werden. Dieser Dialog ist bereits Teil der SMEEX Standard Implementation, nicht mehr der Demo-Applikation. Man kann sowohl bestehenden Profile ändern, als auch neue Profile anlagen (Menü Datei → Neues Profil erstellen).
Der Dialog öffnet sich nach einem Klick auf den Button „Profile“ im Hauptdialog.

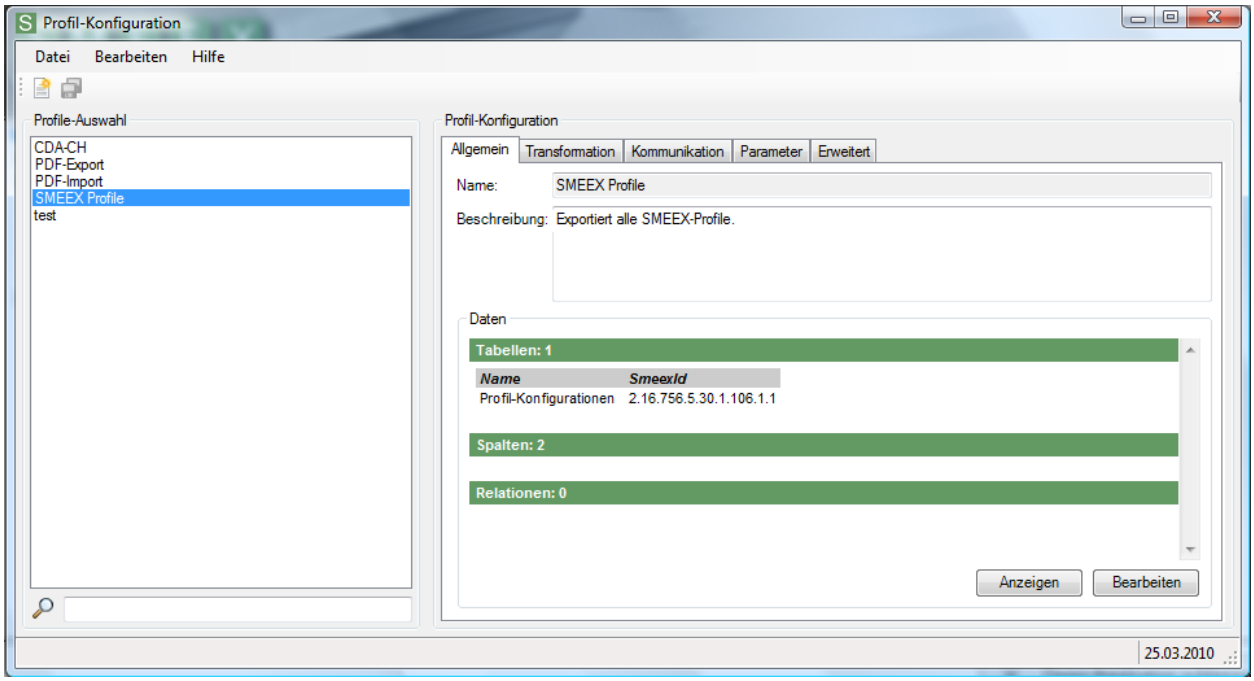

Abbildung 30: Demo-Applikation: Profil-Konfiguration, Tab-Register „Allgemein“

Die im Archiv enthaltenen Datenfelder, die über das ausgewählte Profil definiert werden, können angezeigt und bearbeitet werden (siehe 4.4.2.2). Über den Button „Anzeigen“ gelangt man in den Archiv-Viewer. Im Archiv Viewer werden die Eigenschaften der im linken Baum ausgewählten Objekte (z.B. der Datenfelder unter „Tabellen“), die Daten und Verknüpfungen der ausgewählten Tabelle auf andere im Archiv enthaltenen Tabellen in den jeweiligen Tab-Registern angezeigt.

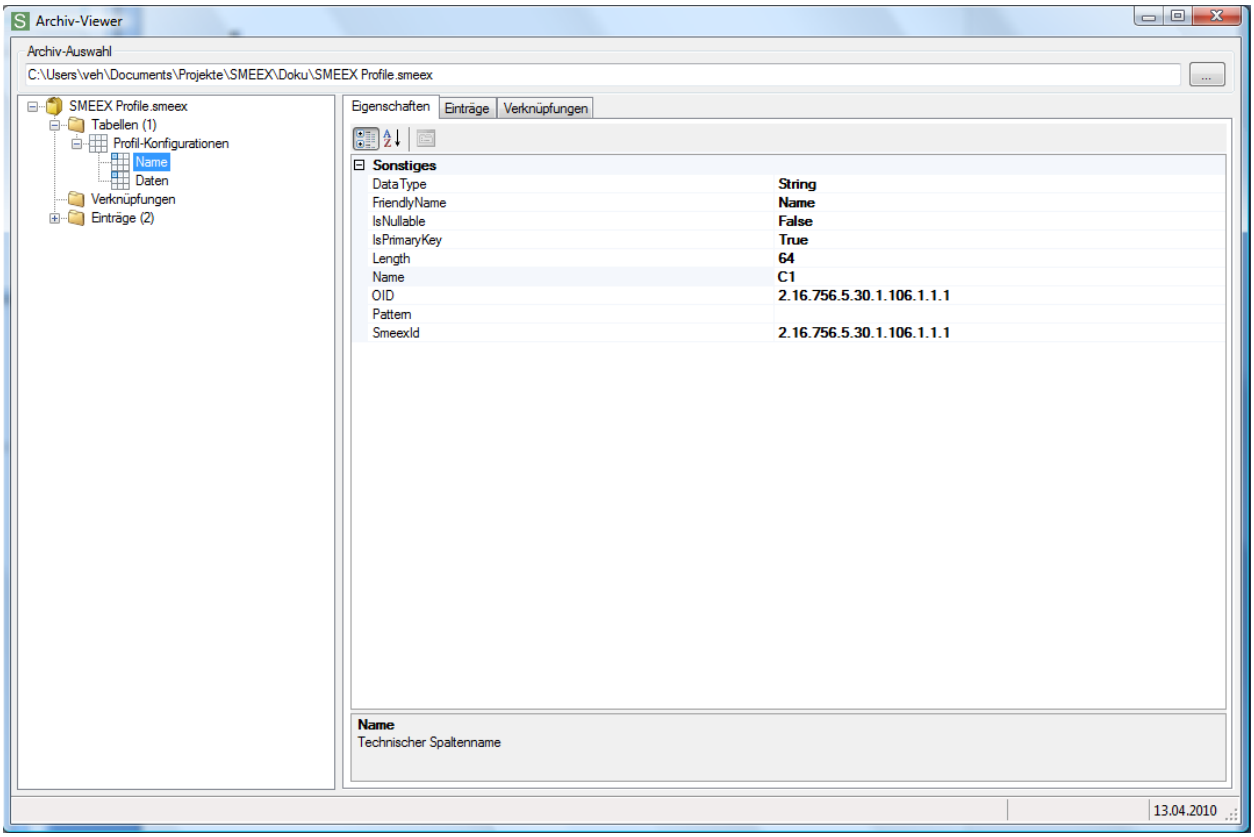

Abbildung 31: Demo-Applikation: Archiv Viewer, Tab-Register „Eigenschaften“

Werden die Eigenschaften einer Verknüpfung angezeigt, so können im Column-Auflistungs-Editor die beteiligten Spalten des Primär- bzw. Fremd-Schlüssels angezeigt werden (Zeile markieren, dann erschient ein Button „…“).

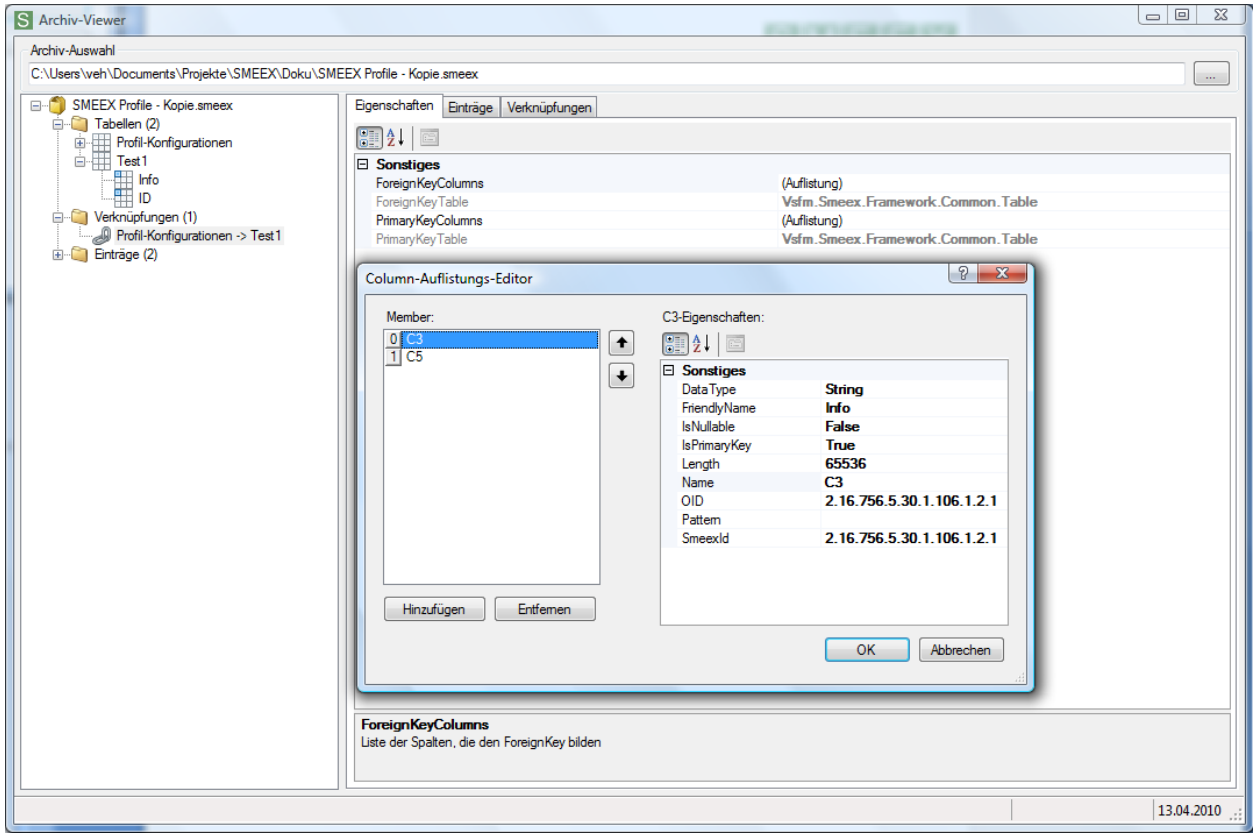

Abbildung 32: Demo-Applikation: Archiv Viewer, Column-Auflistungs-Editor

Im Tab-Register Daten werden die Inhalte der im SMEEX Archiv enthaltenen Files angezeigt. dazu muss im Baum unter dem Ordner „Einträge“ das gewünschte Objekt im Archiv ausgewählt werden. Das gilt nicht nur für die Datenfiles (Darstellung analog Metadaten-Datei), sondern auch für die Metadaten-Files (siehe Abbildung 33) und binäre Files, sofern sie in einem Browser dargestellt werden können (z.B. PDF, JPG, GIF, MS Office Files je nach Konfiguration im Browser oder in separat gestarteten MS Office Instanz).

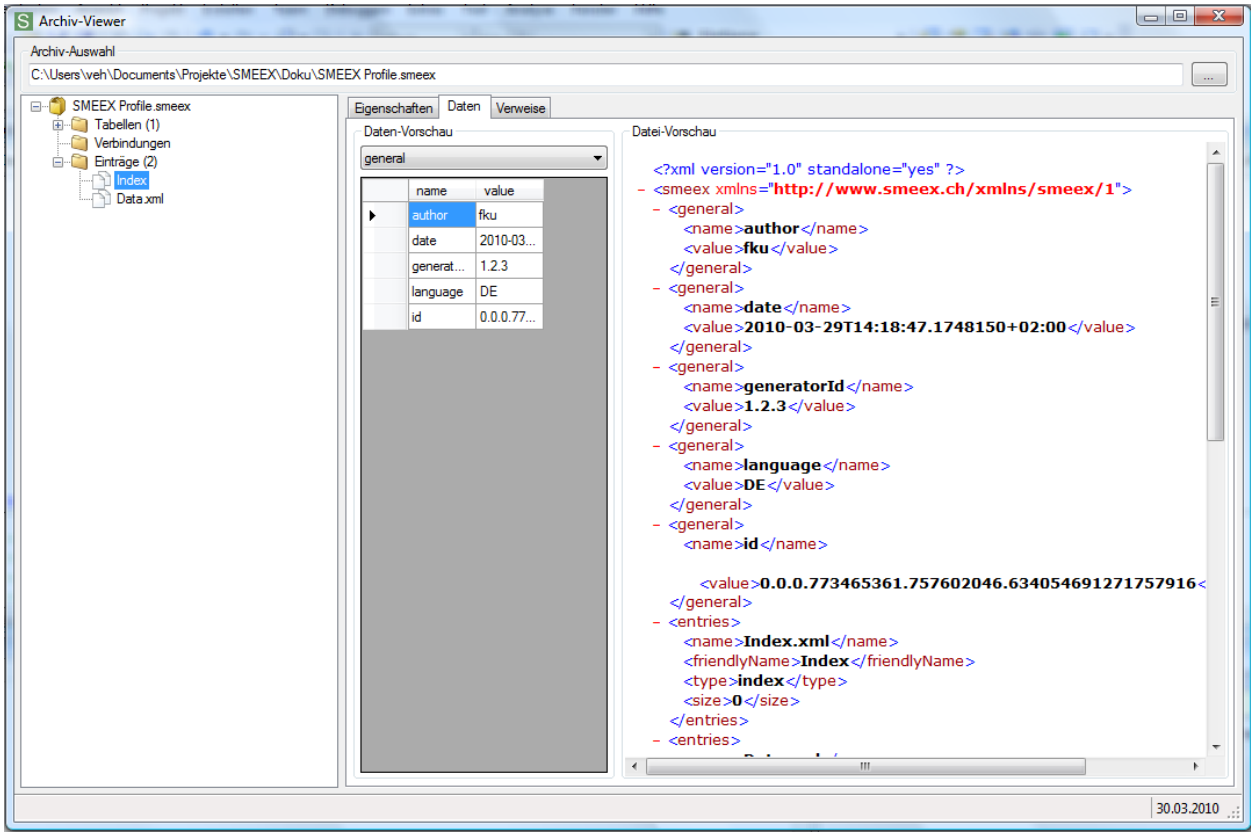

Abbildung 33: Demo-Applikation: Archiv Viewer, Darstellung Metadaten-File

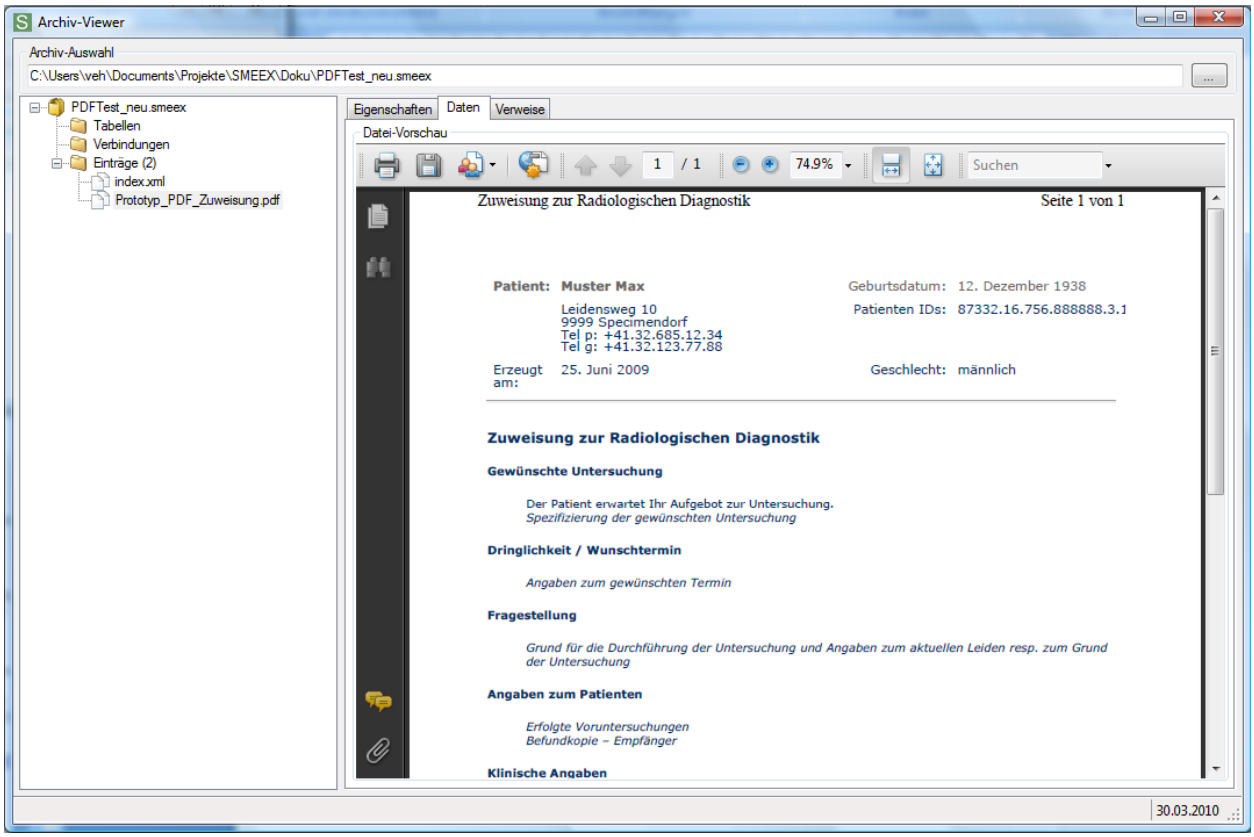

Abbildung 34: Demo-Applikation: Profil Konfiguration, Darstellung binäres File (PDF)

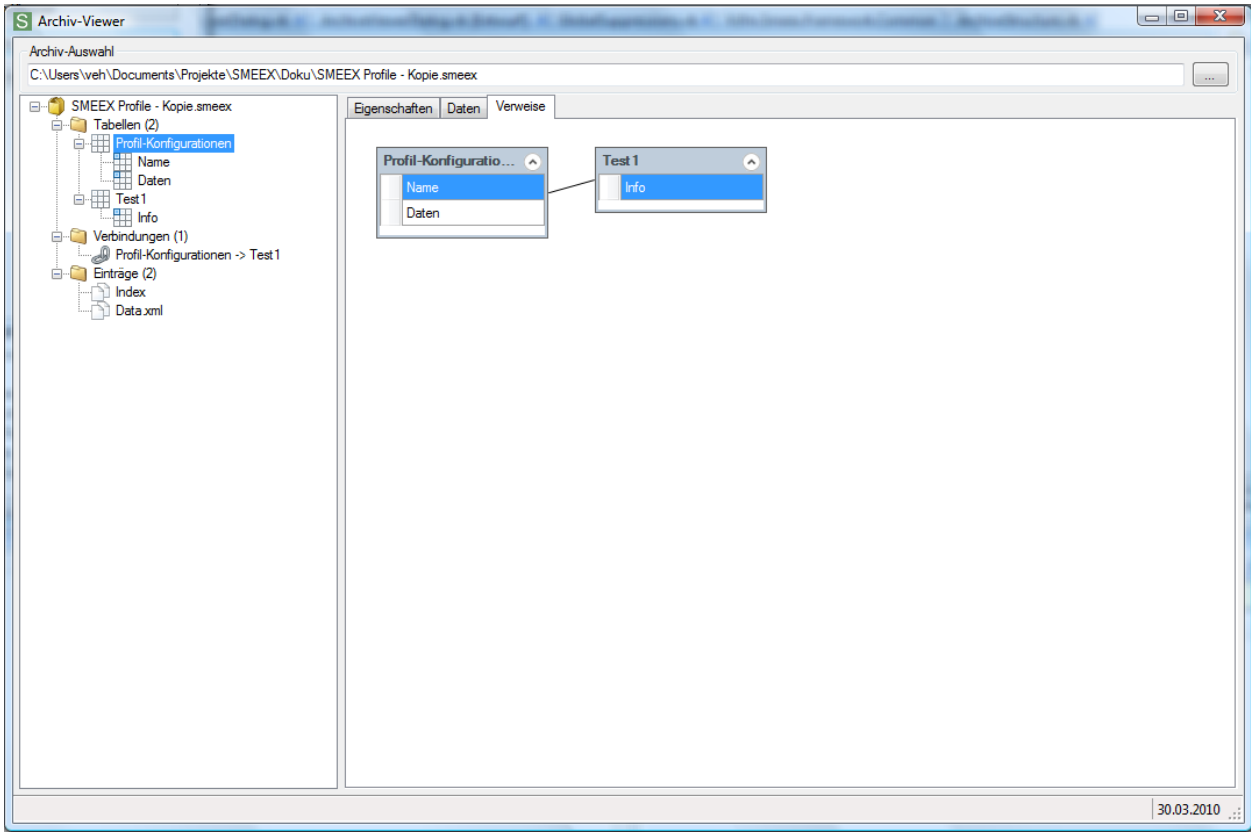

Abbildung 35: Demo-Applikation: Archiv Viewer, Tab-Register „Verweise“

Über den Button „Bearbeiten“ gelangt man in den Auswahldialog, in dem ein im smeexIDManager (siehe 4.6) definiertes Daten-Profil ausgewählt werden kann.

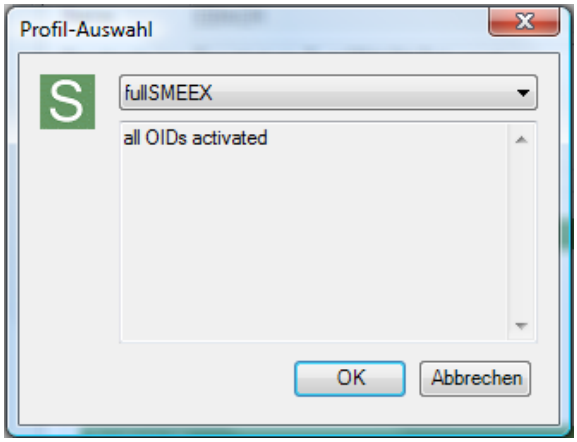

Abbildung 36: Demo-Applikation: Profil Konfiguration: smeexID Daten-Profil Auswahl

Auf den Tab-Register „Transformation“ und „Kommunikation“ in der Profil Konfiguration können die entsprechenden PlugIns dem Profil zugeordnet und konfiguriert werden (siehe 4.4.2.4).

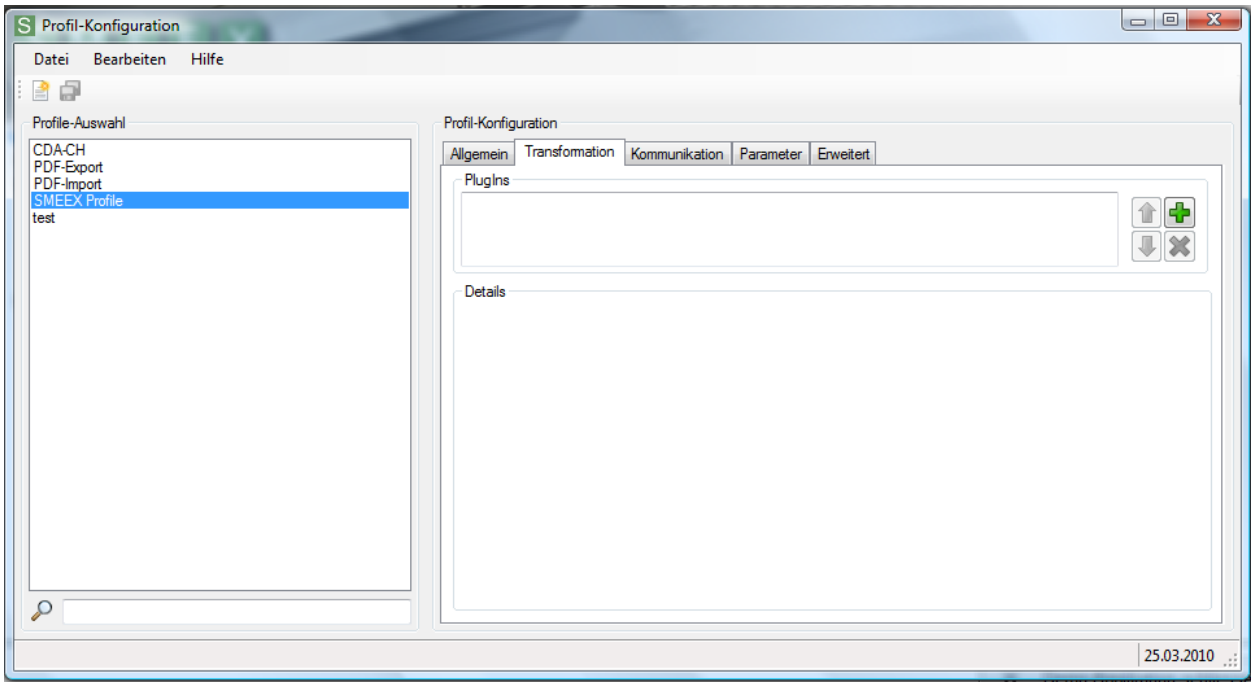

Abbildung 37: Demo-Applikation: Profil-Konfiguration, Tab-Register „Transformation“

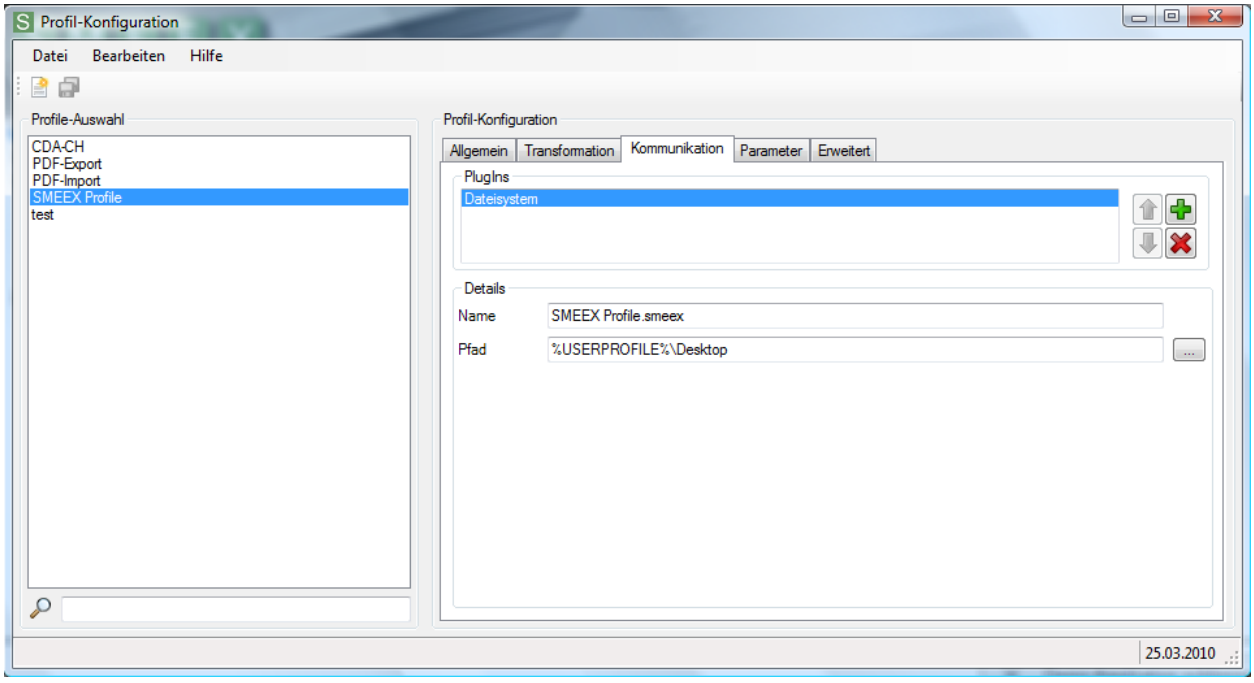

Abbildung 38: Demo-Applikation: Profil-Konfiguration, Tab-Register „Kommunikation“

Auf dem Tab-Register „Parameter“ können die dynamischen Daten, die der Applikation übergeben werden sollen, konfiguriert werden (siehe 4.4.2.3).

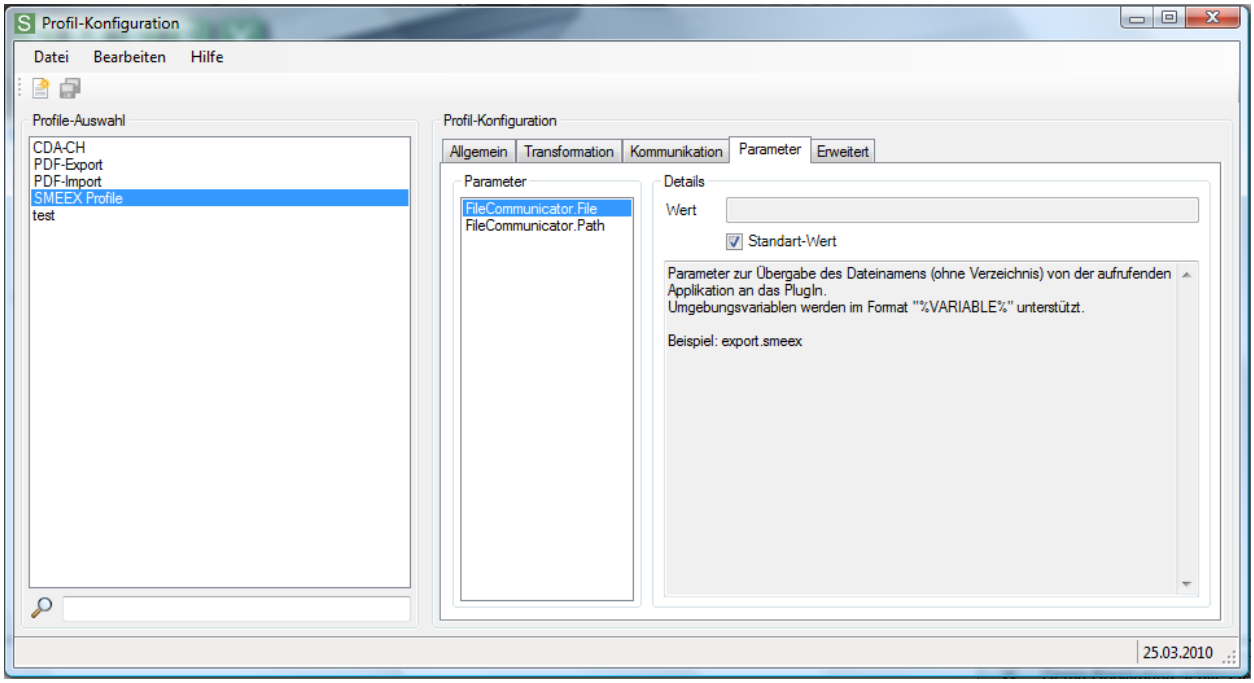

Abbildung 39: Demo-Applikation: Profil-Konfiguration, Tab-Register „Parameter“

Auf dem Tab-Register „Erweitert“ können dem Profil PlugIns für Export und Import zugeordnet werden (Buttons „…“) (siehe 4.4.2.4).

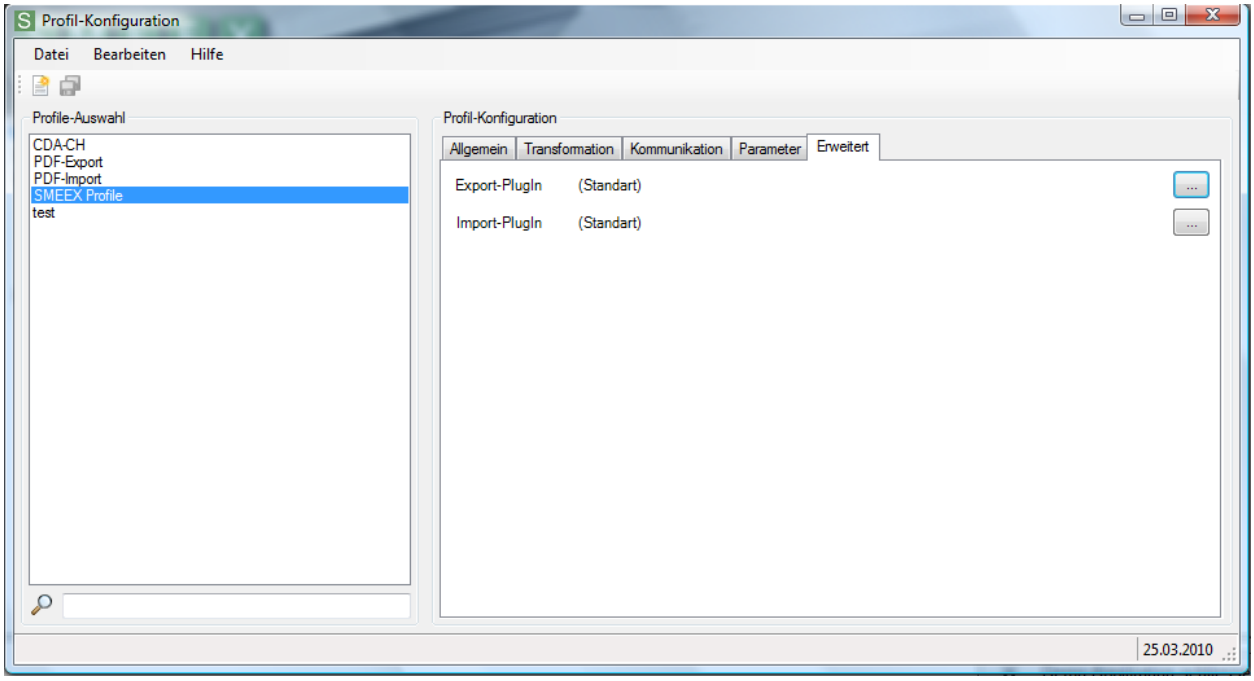

Abbildung 40: Demo-Applikation: Profil-Konfiguration, Tab-Register „Erweitert“

4.8.5 Anzeige des VBA-Makro-Codes

Der Aufruf der SMEEX Framework Wrapper Klasse in der zur Demo Applikation gehörenden Bibliothek *Vsfm.Smeex.DemoDB.Integration.dll* (siehe Abbildung 25) befindet sich in einem VBA Makro.
Um den Makro Code anzuzeigen, muss der in MS Access enthaltene Visual Basic-Editor gestartet werden (Menü Datenbanktools → Visual Basic).

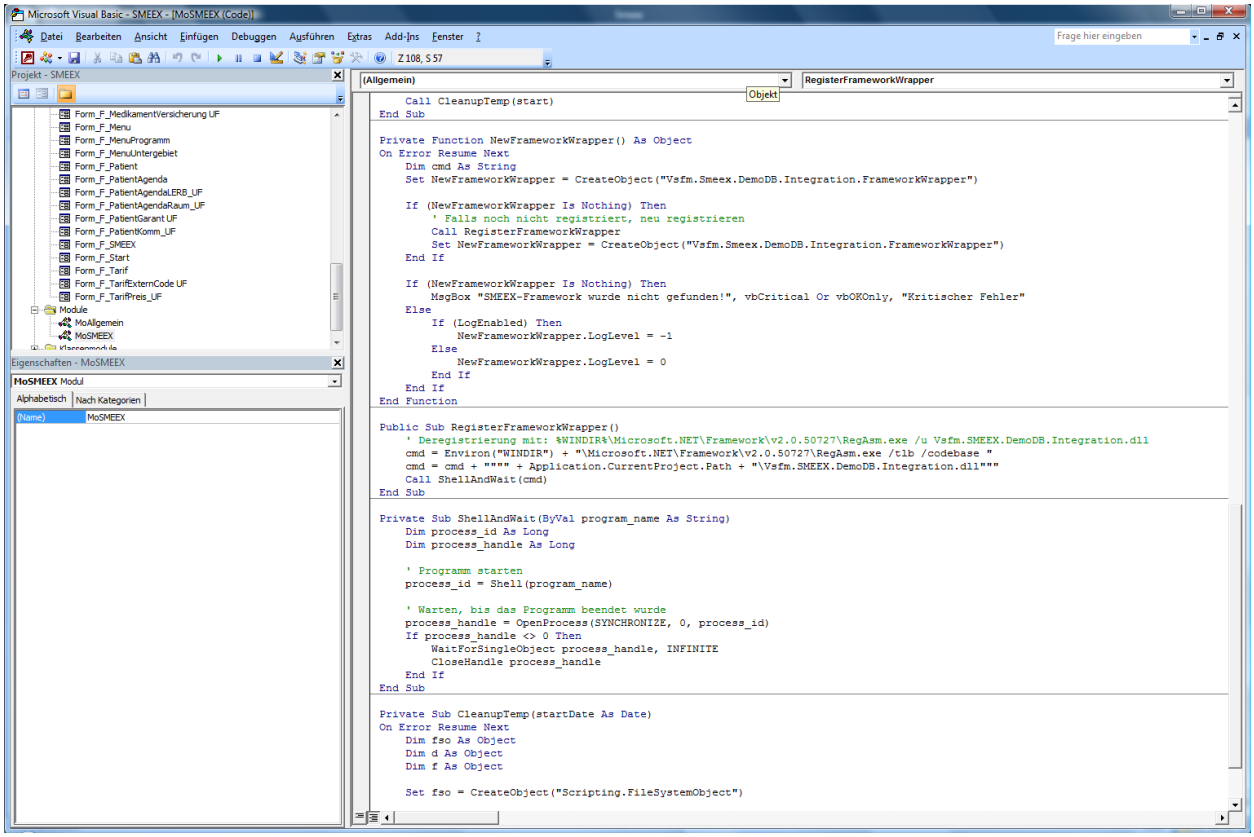

Abbildung 41: Demo-Applikation: Anzeige des Makro-Codes

4.9 UCUM Bibliothek

4.9.1 Motivation

Zu den Daten im medizinischen Bereich gehören auch Messergebnisse aus Laboruntersuchungen in verschiedensten Masseinheiten. Für Konvertierungen und Datenexporte benötigt man also auch hier einen einheitlichen Standard. VSFM stellt aus diesem Grund eine Implementierung des UCUM Systems zur Verfügung. Die vom VSFM zur Verfügung gestellten Standardimplementationen des Frameworks und von PlugIns werden diese Bibliothek für die Validierung und Konvertierung von Einheiten nutzen.

4.9.2 Der UCUM Standard

Der **U**nified **C**ode for **U**nits of **M**easure (UCUM) ist ein regelbasiertes Kodierungssystem für Masseinheiten, dass den elektronische Datenaustausch von physikalischen Grössen vereinfachen soll. Das System umfasst, Prefixes (z.B. k für Kilo), Basiseinheiten (z.B. m für Meter) und abgeleitete Einheiten (z.B. l für Liter). Die formale Definition umfasst den Code jeder Einheit (z.B. m für Meter), das Drucksymbol, den Namen und eine Beschreibung auf Englisch, die Zuordnung zu einer Klasse von Einheiten (z.B. SI-Einheiten) und im Fall der abgeleiteten Einheiten eine Umrechnungsvorschrift in Basiseinheiten.

Die formale Definition, Codetabellen und Beispielimplementierungen finden sich auf den UCUM-Seiten des Regenstrief Institute (http://www.regenstrief.org).

Die vorliegende UCUM Bibliothek ist auf Basis einer bestehenden Java Bibliothek des Open Healthcare Framework (OHF) entstanden (Repository mit Sourcen siehe 7.2.2). Die Java Bibliothek wurde auf .NET portiert und teilweise substantiell umgebaut.
Folgende Umbauten wurden vorgenommen:

- Anpassung der Schnittstelle *IUcumService* an die Bedürfnisse des SMEEX Frameworks, dies betrifft den Umfang sowie die Funktionalität der zur Verfügung gestellten Methoden
- Neuimplementierung der (internen) Klasse *Converter*, die die Konvertierung der angegebenen (zusammengesetzten) Einheit in ihre kanonische Form durchführt Beispiel: für Geschwindigkeiten ist die kanonische Form m.s-1 (Meter mal Sekunde hoch minus eins)
- Erweiterung des Parsers durch die Erkennung der Klammerung von Ausdrücken
- Implementierung von Handlern für die Spezialeinheiten Grad Celsius und Grad Fahrenheit
- Anpassung der impliziten Klammerung bei Divisionen in zusammengesetzten Einheiten an die UCUM Spezifikation
- Erweiterung des Exception Handlings

In der Bibliothek wird zum einen eine allgemeine Schnittstelle für Einheiten-Konvertierung und -Validierung zur Verfügung gestellt. Ebenso steht eine Implementierung zur Verfügung, die optional genutzt werden kann.
Die beiliegende Demo Applikation demonstriert die Nutzung der Bibliothek, der Source Code der Demo-Applikation liegt bei.

4.9.3 Schnittstelle

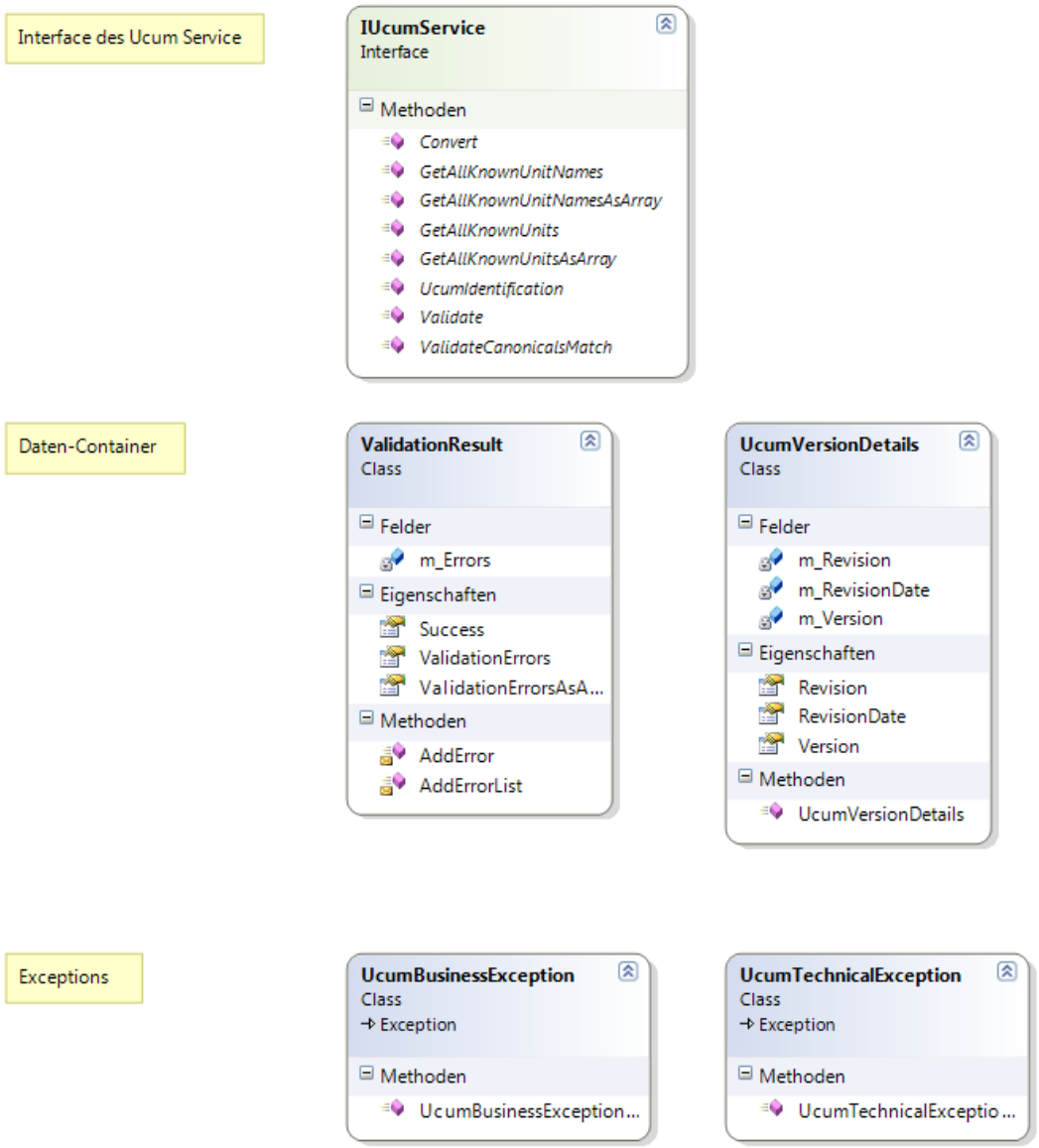

Abbildung 42: Schnittstelle der UCUM Klassenbibliothek

Signatur	Beschreibung
`UcumVersionDetails UcumIdentification()`	gibt Details zur benutzten UCUM Definition zurück
`ValidationResult Validate(String unit)`	überprüft, ob der übergebene Code eine gültige Einheit darstellt
`ValidationResult ValidateCanonicalsMatch(String unit1, String unit2)`	prüft, ob die beiden übergebenen Einheiten eine gemeinsame kanonische Form besitzen und damit in einander umrechenbar sind; implizit wird auch geprüft, ob es sich bei beiden Codes um gültige Bezeichner für Einheiten handelt.

Signatur	Beschreibung
`Decimal Convert(Decimal value, String sourceUnit, String destUnit)`	konvertiert den übergebenen Wert von einer Einheit in die andere
`List<String> GetAllKnownUnits()`	gibt eine Liste aller in der benutzen UCUM Definition bekannten Einheiten (Basiseinheiten und abgeleitete Einheiten) zurück
`String[] GetAllKnownUnitsAsArray();`	analog *GetAllKnownUnits*(), für COM Interop
`List<String> GetAllKnownUnitNames()`	gibt eine Liste der Namen aller in der benutzen UCUM Definition bekannten Einheiten (Basiseinheiten und abgeleitete Einheiten) zurück (Englisch)
`String[] GetAllKnownUnitNamesAsArray();`	analog *GetAllKnownUnitNames*(), für COM Interop

Tabelle 15: Methoden der UCUM Schnittstelle IUcumService

Exception	Beschreibung
UcumTechnicalException	Es ist ein technischer Fehler aufgetreten. Der Benutzer kann nichts zur Behebung des Fehlers tun.
UcumBusinessException	Es ist ein Business Fehler aufgetreten. Der Benutzer kann durch anderes Verhalten (z.B. andere Eingaben) den Fehler vermeiden.

Tabelle 16: Exceptions der UCUM Schnittstelle

4.9.4 Demo-Applikation

Die Demo-Applikation zeigt die Nutzung der UCUM Bibliothek und demonstriert ihre gesamte Funktionalität. Im Wesentlichen handelt es sich dabei um den Aufruf und Nutzung einer Instanz der in der Bibliothek enthaltenen Implementierung von *IUcumService*: *UcumEssenceService*.

Die UCUM Demo-Applikation ist eine Windows Anwendung und steht als ausführbare Datei (.exe) zur Verfügung. Da der Sourcecode zur Verfügung gestellt wird, kann sie auch aus der Entwicklungsumgebung heraus gestartet werden.

Die Auswahlfelder für Einheiten werden über die Methode *GetAllKnownUnits* befüllt.

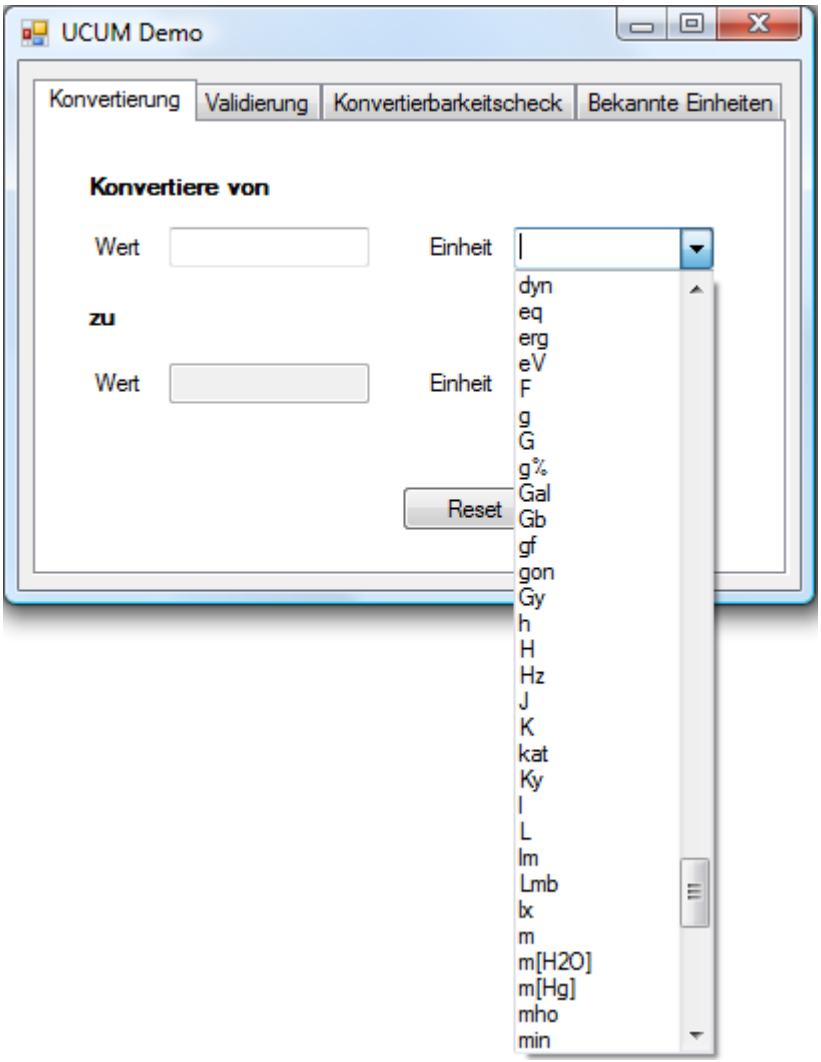

Abbildung 43: UCUM Demo-Applikation: Auswahlfeld für Einheiten

Im Tab-Register „Konvertierung“ kann eine Einheit in eine andere konvertiert werden.

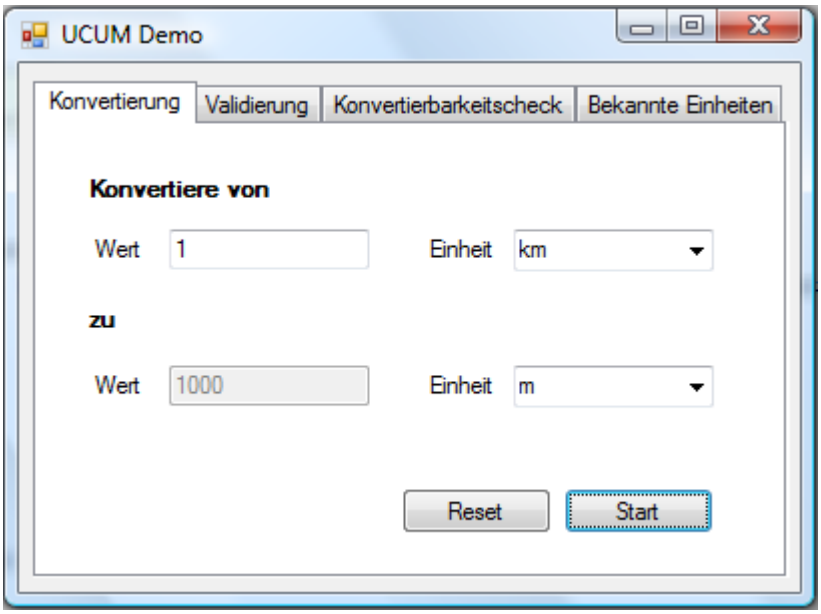

Abbildung 44: UCUM Demo-Applikation: Tab-Register "Konvertierung"

Im Tab-Register „Validierung“ können Einheiten auf ihre Gültigkeit überprüft werden.

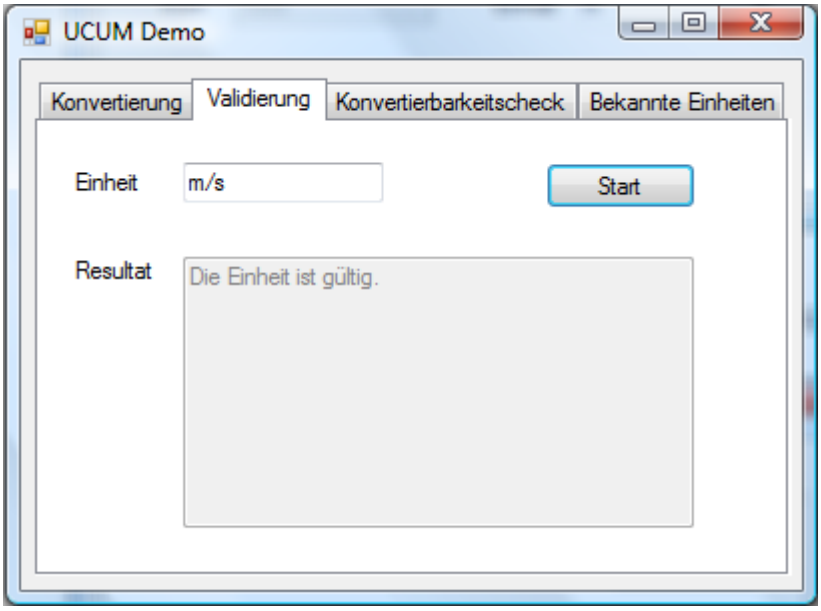

Abbildung 45: UCUM Demo-Applikation: Tab-Register "Validierung"

Im Tab-Register „Konvertierbarkeitscheck“ werden zwei Einheiten daraufhin überprüft, ob sie in einander umrechenbar sind.

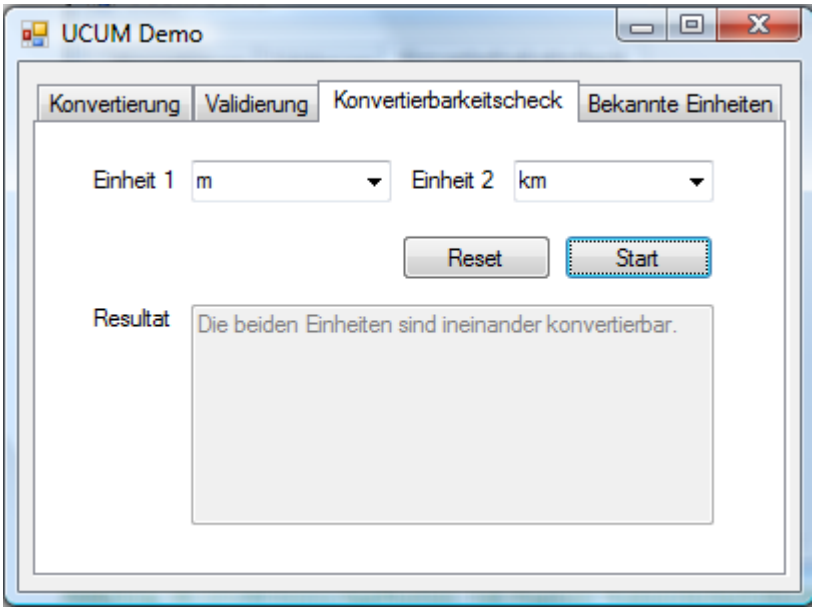

Abbildung 46: UCUM Demo-Applikation: Tab-Register "Konvertierbarkeitscheck"

Im Tab-Register wird eine Liste der Namen der bekannten Einheiten angezeigt.

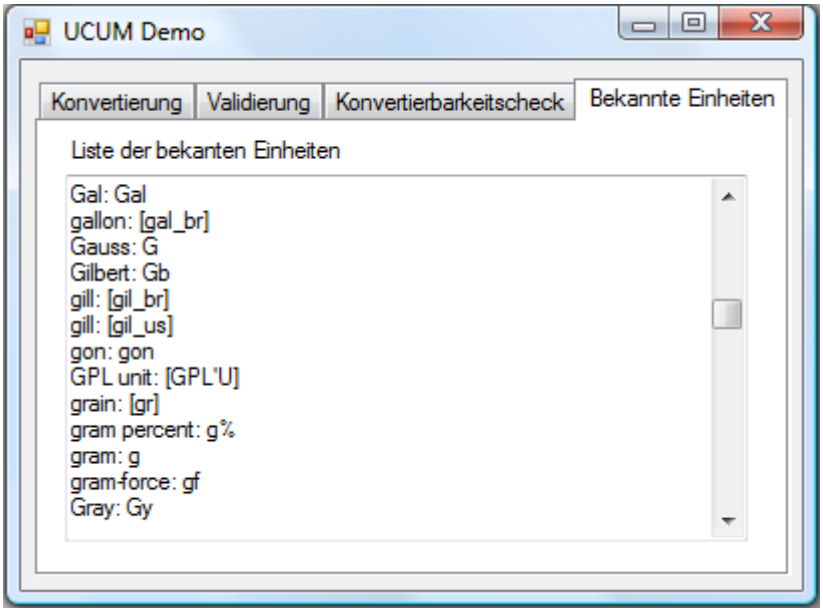

Abbildung 47: UCUM Demo-Applikation: Tab-Register "Bekannte Einheiten"

4.10 Technische Verifikation von SMEEX Archiven

4.10.1 Überblick

Der SMEEX Datenaustausch-Standard soll dazu beitragen, dass Daten (Gesundheitswesen Schweiz) auf einfache Art und Weise über die Systemgrenzen hinweg ausgetauscht werden können.
Die Branche und letzten Endes der Anwender erlangt nur Sicherheit im Thema Datenaustausch, wenn die notwendigen Applikationsfunktionen umfangreich zur Verfügung stehen und eine entsprechende Zertifizierung durch den Verband (VSFM) gemacht wird.

In einem ersten Schritt wird den Herstellern von Branchen-Software eine technische Verifikation ihrer Implementierung des SMEEX Standards angeboten. Konkret handelt es sich dabei um eine Überprüfung der technischen Struktur von SMEEX Archiven. Hersteller können von ihrer Applikation erzeugte SMEEX Archive beim Verband einreichen. Diese Archive werden dann auf strukturelle Einhaltung des SMEEX Standards geprüft und das Ergebnis dem Hersteller mitgeteilt.

Die Verifikation wird von VSFM als kostenloser Service angeboten.

Zu einem späteren Zeitpunkt soll es die Möglichkeit geben, für Produkte eine offizielle Zertifizierung der SMEEX-Konformität zu erhalten.

4.10.2 Prozess

Auf der Homepage der SMEEX Initiative (http://www.smeex.ch) ist eine eigene Seite geplant, die der technischen Verifikation gewidmet ist. Sie wird auf der Einstiegsseite über einen Link im Hauptnavigationsmenü erreichbar sein.

Auf dieser Seite wird der Prozess der technischen Verifikation beschrieben, inklusive einer Auflistung der Testinhalte. Eine Anleitung gibt die Schritte an, die für die Verifikation nötig sein werden.

Nach erfolgter Verifikation wird eine Rückmeldung über das Resultat gegeben. Sie enthält eine Referenz auf das geprüfte SMEEX Archiv, das Ergebnis der Prüfung (bestanden / nicht bestanden), sowie allenfalls eine Liste der Gründe für einen Misserfolg.

4.10.3 Inhalt der Prüfung

Im Rahmen der technischen Verifikation werden die folgenden Punkte überprüft:

- File Struktur des exportierten Archivs entspricht dem SMEEX Standard
- Das Metadatenverzeichnis ist mit Hilfe des Schemas validierbar
- Inhalt und Umfang des exportierten Archivs entsprechen dem SMEEX Standard

5 Literaturverzeichnis

Für eine vertiefte Auseinandersetzung mit den Themen elektronischer Datenaustausch im Gesundheitswesen und Klassifizierungssysteme ist in diesem Verzeichnis Hintergrundliteratur aufgeführt. Die Liste ist nicht vollständig oder erschöpfend, sie hat lediglich Beispiel-Charakter. Es existieren noch eine Vielzahl weitere Publikationen zu diesen Themenbereichen, deren Auflistung aber den Rahmen sprengen würde.
Neben den aufgeführten Publikationen ist auf den Homepages der mit eHealth befassten Organisationen vielfältiges Material zu finden (siehe 7.2.1).

Tim Benson: *Principles of Health Interoperatbility HL7 and SNOMED*, Health Informatics Series, Springer Verlag

Achim Jäckel (Hrsg.): *Telemedizinführer Deutschland Ausgabe2009*, Deutsches Medizin Forum

Clem McDonald, MD, Stan Huff, MD, Kathy Mercer, Jo Anna Hernandez,
Daniel J. Vreeman, PT, DPT (Ed.): *Logical Observation Identifiers Names and Codes (LOINC®) Users' Guide*, LOINC-Projekt des Regenstrief Institutes, Bezugsquelle: http://www.loinc.org

Bundesamt für Gesundheit: *Strategie eHealth*, Bezugsquelle: http://www.e-health-suisse.ch/hinweise/index.html?lang=de
An dieser Stelle sind weitere Dokumente über die eHealth Strategie des Bundes erhältlich.

Marcel Hanselmann, Christoph Knoepfel, Tony Schaller, Peter Steiner: *CDA-CH: SPEZIFIKATION ZUM ELEKTRONISCHEN AUSTAUSCH VON MEDIZINISCHEN DOKUMENTEN IN DER SCHWEIZ*, HL7 Benutzergruppe Schweiz, Bezugsquelle: http://www.hl7.ch/publikationen0.html

Quellen des vorliegenden Dokuments:
Diverse interne Dokumente des VSFM der Autoren Reto Mettler, Freddy Kühne und Stefan Gerber.

6 Autoren

Reto Mettler
dipl. Ing. FH et EMBA

Studium der Systemtechnik und der Unternehmensführung in Buchs und St. Gallen. 1999-2002 als Software-Ingenieur für Entwicklung der elektronischen Krankengeschichte für Zahnärzte (StomaNet) verantwortlich. StomaNet zählt heute zu den am meisten verbreiteten elektronischen Krankengeschichten für Zahnärzte im klinischen Umfeld. 2002 Konzeption und Umsetzung von umfangreichen ICT-Projekten in medizinischen Kliniken und Instituten. 2003 bis 2008 Bereichsleiter Softwareentwicklung und Mitglied der Geschäftsleitung bei der Vitodata AG in Seuzach (Oberohringen bei Winterthur). Leitend bei der Lancierung und Umsetzung der SMEEX-Initiative (elektronischer Datenaustausch im schweizerischen Gesundheitswesen, SMEEX steht für **S**wiss **ME**dical data **EX**change). In diesem Zusammenhang Autor verschiedener technischer Publikationen (vgl. http://www.smeex.ch/smeex-der-standard/publikationen-detailbeschreibungen/).

2009 Stv. Geschäftsführer der Vitodata AG. Ab 2010 Geschäftsführer der Vitodata AG.

Kontaktadresse
Vitodata AG
Deisrütistrasse 10
8472 Oberohringen bei Winterthur

Phone 052 320 55 55
Email reto.mettler@vitodata.ch

Co Autorin
Viola Ehrensperger

Studium der Mathematik, Physik und Informatik für das Lehramt an Gymnasien in Erlangen (D) und Ashland, Oregon (USA). Ab 2002 als Software-Ingenieurin in Projekten für Kunden in verschiedensten Branchen tätig.
Seit 2009 bei der Vitodata AG für die Wartung und Weiterentwicklung von StomaNet verantwortlich. Mitarbeit im SMEEX-Projekt : Portierung und Refactoring der UCUM Bibliothek und Erstellung des vorliegenden Dokuments.

Kontaktadresse
Vitodata AG
Deisrütistrasse 10
8472 Oberohringen bei Winterthur

Phone 052 320 55 55
Email viola.ehrensperger@vitodata.ch

7 Anhang

7.1 XML-Schema der Archiv-Metadaten

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
           targetNamespace="http://www.vitodata.ch/xmlns/smeex/1"
           xmlns:smeex="http://www.vitodata.ch/xmlns/smeex/1"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- Schema für SMEEX-Metadaten ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
  <xs:element name="smeex">
    <xs:complexType>
      <xs:sequence>
        <!-- Allgemeine Einträge -->
        <xs:element minOccurs="5" maxOccurs="unbounded" name="general">
          <xs:complexType>
            <xs:all>
              <xs:element name="name" type="xs:NCName">
              </xs:element>
              <xs:element name="value" type="xs:string" />
            </xs:all>
          </xs:complexType>
        </xs:element>

        <!-- Dateien im SMEEX-Archiv -->
        <xs:element minOccurs="1" maxOccurs="unbounded" name="files">
          <xs:complexType>
            <xs:all>
              <xs:element name="name" type="xs:NCName" />
              <xs:element name="friendlyName" type="smeex:friendlyNameType" />
              <xs:element name="type" type="smeex:fileTypeType" />
              <xs:element name="size" type="xs:unsignedInt" />
            </xs:all>
          </xs:complexType>
        </xs:element>
```

```
<!-- Tabellen-Definitionen -->
<xs:element minOccurs="1" maxOccurs="unbounded" name="tables">
  <xs:complexType>
    <xs:all>
      <xs:element name="name" type="xs:NCName" />
      <xs:element name="friendlyName" type="smeex:friendlyNameType" />
      <xs:element name="rowCount" type="xs:unsignedInt" />
      <xs:element name="smeexId" type="smeex:smeexIdType" />
    </xs:all>
  </xs:complexType>
</xs:element>

<!-- Spalten-Definitionen -->
<xs:element minOccurs="1" maxOccurs="unbounded" name="columns">
  <xs:complexType>
    <xs:all>
      <xs:element name="table" type="xs:NCName" />
      <xs:element name="name" type="xs:NCName" />
      <xs:element name="friendlyName" type="smeex:friendlyNameType" />
      <xs:element name="type" type="smeex:typeType" />
      <xs:element name="length" type="xs:unsignedInt" />
      <xs:element name="isPrimaryKey" type="xs:boolean" />
      <xs:element name="isNullable" type="xs:boolean" />
      <xs:element name="smeexId" type="smeex:smeexIdType" />
    </xs:all>
  </xs:complexType>
</xs:element>

<!-- Tabellen-Relationen -->
<xs:element minOccurs="0" maxOccurs="unbounded" name="relations">
  <xs:complexType>
    <xs:all>
      <xs:element name="primaryKeyTable" type="xs:NCName" />
      <xs:element name="primaryKeyColumns" type="xs:string" />
      <xs:element name="foreignKeyTable" type="xs:NCName" />
      <xs:element name="foreignKeyColumns" type="xs:string" />
    </xs:all>
  </xs:complexType>
```

```
      </xs:element>
    </xs:sequence>
  </xs:complexType>

  <!-- Unique-Einschränkungen ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
  <!-- Tabelle "general" -->
  <xs:unique name="generalName">
    <xs:selector xpath="smeex:general" />
    <xs:field xpath="smeex:name" />
  </xs:unique>

  <!-- Tabelle "files" -->
  <xs:unique name="filesName">
    <xs:selector xpath="smeex:files" />
    <xs:field xpath="smeex:name" />
  </xs:unique>

  <!-- Tabelle "tables" -->
  <xs:unique name="tablesName">
    <xs:selector xpath="smeex:tables" />
    <xs:field xpath="smeex:name" />
  </xs:unique>

  <!-- Tabelle "columns" -->
  <xs:unique name="columnsName">
    <xs:selector xpath="smeex:columns" />
    <xs:field xpath="smeex:table" />
    <xs:field xpath="smeex:name" />
  </xs:unique>

  <!-- Tabelle "relations" -->
  <xs:unique name="relationsTable">
    <xs:selector xpath="smeex:relations" />
    <xs:field xpath="smeex:primaryKeyTable" />
    <xs:field xpath="smeex:foreignKeyTable" />
    <xs:field xpath="smeex:foreignKeyColumns" />
  </xs:unique>
</xs:element>
```

```
<!-- Typen-Deklarationen ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Dateityp-Deklaration -->
<xs:simpleType name="fileTypeType" id="fileTypeType">
  <xs:restriction base ="xs:string">
    <xs:enumeration value="index" />
    <xs:enumeration value="data" />
    <xs:enumeration value="annex" />
  </xs:restriction>
</xs:simpleType>

<!-- friendlyName-Deklaration -->
<xs:simpleType name="friendlyNameType" id="friendlyNameType">
  <xs:restriction base ="xs:NCName">
    <xs:maxLength value="32" />
  </xs:restriction>
</xs:simpleType>

<!-- SmeexId-Deklaration -->
<xs:simpleType name="smeexIdType" id="smeexIdType">
  <xs:restriction base ="xs:string">
    <xs:pattern value="(\d*\.)+\d*"/>
  </xs:restriction>
</xs:simpleType>

<!-- Deklaration der unterstützen XSD-Typen -->
<xs:simpleType name="typeType" id="typeType">
  <xs:restriction base ="xs:string">
    <xs:enumeration value="base64Binary" />
    <xs:enumeration value="boolean" />
    <xs:enumeration value="byte" />
    <xs:enumeration value="datetime" />
    <xs:enumeration value="decimal" />
    <xs:enumeration value="double" />
    <xs:enumeration value="float" />
    <xs:enumeration value="int" />
    <xs:enumeration value="long" />
    <xs:enumeration value="short" />
```

```
      <xs:enumeration value="string" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

7.2 Link-Sammlung

Die Einträge der Link Sammlung sind unsortiert. Ihre Reihenfolge stellt keine Wertung ihrer Relevanz dar.

7.2.1 Organisationen und Standards im Gesundheitswesen

Homepage des SMEEX-Standards
http://www.smeex.ch

Verband Schweizerischer Fachhäuser für Medizinal-Informatik
http://www.vsfm.ch

Homepage des swiss medical reference systems
http://www.smerf.ch

Koordinationsorgan eHealth Bund-Kantone
http://www.e-health-suisse.ch/

Interessengemeinschaft eHealth
http://www.ig-ehealth.ch

Homepage des HL7 Standards und der HL7 Benutzergruppe Schweiz
http://www.hl7.ch
HL7 International
http://www.hl7.org

Homepage des Vereins eCH
http://www.ech.ch

Homepage des ICPC-2 Standards
http://www.icpc.ch

Homepage des LOINC Standards
http://www.loinc.org

Schweizerische Gesellschaft für Allgemein Medizin
http://www.sgam.ch

Schweizer, europäische und internationale Homepage der IHE Initiative
http://www.ihe-suisse.ch
http://www.ihe-europe.net
http://www.ihe.net

Homepage des DICOM Standards
http://medical.nema.org/dicom

Homepage des Regenstrief Institutes
http://www.regenstrief.org

Homepage des UCUM Standards
http://unitsofmeasure.org

Verband der Hersteller von IT-Lösungen für das Gesundheitswesen e.V. (D)
http://www.vhitg.de

International Health Terminology Standard Development Organization, pflegt SNOMED CT (**S**ystematized **No**menclature of **Med**icine-**C**linical **T**erms), eine Version der SNOMED
http://www.ihtsdo.org/
Seite mit Informationen zu SNOMED CT
http://www.ihtsdo.org/snomed-ct/

US National Library of Medicine, Liste der Publikationen zu UMLS
http://www.nlm.nih.gov/pubs/factsheets/factsheets.html#A8

Bundesamt für Gesundheit BAG
http://www.bag.admin.ch/

7.2.2 Entwickler-Ressourcen

BeispielProjekt für den Aufruf von .NET Objekten aus einem Java Projekt:
http://www.microsoft.com/germany/msdn/solve/codeclips/library.aspx?id=msdn_de_36042

Mono Projekt-Homepage
http://www.mono-project.com

Projekt-Homepage des OHF
http://www.eclipse.org/ohf
Repository des OHF UCUM Projekts
http://dev.eclipse.org/viewcvs/index.cgi/org.eclipse.ohf/plugins/org.eclipse.ohf.h3et.ucum/?root=Technology_Project)

7.2.3 Sonstige

Internationales OID Repository. Hier ist die Schweizer Root-OID (2.16.756.5) registriert.
http://www.oid-info.com
Schweizer OID Repository, zeigt auf die OID-Seite der HL7 Benutzergruppe (http://www.hl7.ch/oid).
http://www.oid-register.ch

Bundesamt für Kommunikation
http://www.bakom.ch

Schweizer Markenregister
http://www.swissreg.ch

Eidgenössisches Institut für Geistiges Eigentum (Informationen zu Marken)
http://www.ige.ch

Freie Enzyklopädie
http://www.wikipedia.ch

7.3 Hilfe-Files

Liste der enthaltenen Hilfe-Files.

Hilfe-File	Beschreibung
Vsfm.Smeex.Framework.chm	Hilfefile des Frameworks selbst
Vsfm.Smeex.DemoDB.Integration.chm	Hilfefile der Demo-Applikation