

*xcentric technology & consulting
Oberwiesenstrasse 5
CH - 8057 Zürich*

*+41 (0)43 255 01 69
www.xcentric.ch
info@xcentric.ch*

JAXFront[®] – Konzepte

V 1.5







XML RENDERING TECHNOLOGY

JAXFRONT
www.jaxfront.com

<http://www.jaxfront.com>
jaxfront@xcentric.ch

Dokument-Verwaltung

Autor(en)	C.Gantenbein, M.Leber,S.Portmann info@xcentric.ch					
Version	1.50					
Link	http://www.jaxfront.com/download/jaxfront-konzepte-1_50.pdf					
Klassifizierung	<input checked="" type="checkbox"/> nicht klassifiziert	<input type="checkbox"/> Intern	<input type="checkbox"/> vertraulich	<input type="checkbox"/> geheim		
Bearbeitungsstand	<input type="checkbox"/> Entwurf / in Bearbeitung	<input type="checkbox"/> zur Abnahme	<input checked="" type="checkbox"/> definitive Fassung	<input type="checkbox"/>		

Bild	Beschreibung
	Zeigt eine Beispielsyntax in XML.
	Zeigt ein Codebeispiel in Java.
	Weist auf eine wichtige Information hin.
	Erläutert ein vorher präsentiertes Beispiel.

INHALTSVERZEICHNIS

1	EINFÜHRUNG	5
1.1	AUSGANGSLAGE.....	5
1.2	POSITIONIERUNG.....	6
1.3	DATEN UND IHRE REPRÄSENTATION	7
1.4	DAS BEARBEITEN VON XML-DOKUMENTEN.....	7
1.5	ERSTELLEN VON BENUTZEROBERFLÄCHEN.....	8
1.5.1	<i>Statische vs. dynamische Erzeugung</i>	8
1.5.2	<i>Verwendung von XSLT</i>	9
1.6	DYNAMISCHE GUI GENERIERUNG	10
2	ARCHITEKTUR	11
2.1	MODELL BASIERTE ARCHITEKTUR	11
2.2	KONZEPTIONELLES MODELL.....	12
2.3	BUSINESS MODELL – GUI MODELL	14
2.4	ROLLENVERTEILUNG.....	15
3	CORE RENDERING ENGINE	16
3.1	ÜBERBLICK	16
3.2	DIE 6 PROZESSCHRITTE	17
3.3	PARSING.....	19
3.4	STRUKTUR ANALYSE	21
3.5	DATA BINDING.....	23
3.6	PERSONALISIERUNG DER BENUTZEROBERFLÄCHE.....	25
3.7	RENDERING.....	26
3.8	INTERAKTIVE DATENMANIPULATION, VALIDIERUNG	27
4	SYSTEMKOMPONENTEN	28
4.1	ÜBERBLICK	28
4.2	JAVA SWING RENDERER	29
4.3	HTML RENDERER	30
4.4	PDF RENDERER	31
4.5	XUI EDITOR.....	32
5	SYSTEMINTEGRATION.....	33
5.1	INTEGRATION DES JAVASWING RENDERER.....	34
5.2	INTEGRATION DES HTML RENDERERS.....	36
5.3	INTEGRATION DES PDF RENDERERS.....	38
6	WICHTIGE LINKS.....	39

Abbildungsverzeichnis

Abbildung 1: Positionierung von JAXFront	6
Abbildung 2: Modell basierte Architektur	11
Abbildung 3: JAXFront - Konzeptionelles Modell	12
Abbildung 4: Business Modell - GUI Modell	14
Abbildung 5: JAXFront – Rollenkonzept	15
Abbildung 6: Überblick der Core-Engine	16
Abbildung 7: JAXFront® - Core Rendering Engine	17
Abbildung 8: Ablaufdiagramm der Core Rendering Engine	18
Abbildung 9: W3C DOM-Konzept	19
Abbildung 10: XML Parsertypen	20
Abbildung 11: XMLSchema - Bildung eines Graphen	23
Abbildung 12: Data-Binding an ein JAXFront® DOM	24
Abbildung 13: Verarbeitung der XUI-Komponenten	25
Abbildung 14: Model-View-Controller Paradigma von Java Swing	26
Abbildung 15: Interaktive Datenmanipulation & Validierung	27
Abbildung 16: JAXFront Systemkomponenten	28
Abbildung 17: SwingRenderer (Screenshot)	29
Abbildung 18: HTML Renderer (Screenshot)	30
Abbildung 19: PDF Designer (Screenshot)	31
Abbildung 20: Erzeugtes PDF Dokument	31
Abbildung 21: XUIEditor (Screenshot)	32
Abbildung 22: JAXFront® - 3tier Integration	33
Abbildung 23: HelloWorld für JAXFront® (Codebeispiel)	34
Abbildung 24: Das Benutzen der TypeVisualizerFactory (Codebeispiel)	35
Abbildung 25: JAXFront HTML Architektur	36

1 Einführung

1.1 Ausgangslage

Die steigende Bereitschaft und Akzeptanz, Geschäftsdaten strukturiert in XML zu beschreiben, macht die Darstellung und Nachbearbeitung dieser zu einem wichtigen Faktor in vielen Software Entwicklungsprozessen. XML hat sich als Standard für die Beschreibung von Geschäftsdaten im elektronischen Datenverkehr (EDI) etabliert. Da einer der positiven Eigenschaften von XML ihre Erweiterbarkeit (eXtensibility) ist, stellt dies hohe Anforderungen an die Visualisierung und Nachbearbeitung.

Neue technische Möglichkeiten eröffnen immer effizientere Mechanismen in der Darstellung und der Nachbearbeitung von Daten. Der Anpassungsdruck wird durch die Geschwindigkeit, mit der neue Normen in der IT verabschiedet werden noch weiter verschärft und verlangen somit nach innovativen Konzepten um diesem Umstand gerecht zu werden.

Entscheidend für die generische Darstellung von XML-Dokumenten ist eine exakte Beschreibung der Datenstrukturen, die die Beziehungen und Regeln der darzustellenden Daten definieren.

Für die Generierung von Benutzeroberflächen ist das Vorhandensein einer standardisierten Modellierungssprache ein Muss. Zu diesem Zweck steht die vom W3C standardisierte XML Schemaspezifikation zur Verfügung. Sie expliziert formale syntaktische Anforderungen an XML Dokumente und wird somit zum Schlüsselement im Austausch und Darstellung von XML basierten.

1.2 Positionierung

Geschäftsprozesse, die den Austausch von EDI – Meldungen verlangen, können mit JAXFront® nicht abgebildet werden. Die Kommunikationsschichten zum Austausch von XML Dokumenten und die Integrationsebene, welche für eine reibungslose Integration der erhaltenen Daten in verschiedene Applikationslandschaften sorgt, sind im Zusammenhang mit den hier vorgestellten Konzepten nicht von Relevanz. Die Speicherung von XML Dokumenten (Persistenz) wird als schichtenübergreifende Ebene verstanden und im Rahmen von JAXFront® als Prozess definiert der Datenänderungen in XML Dokumenten persistent macht (serialisiert).

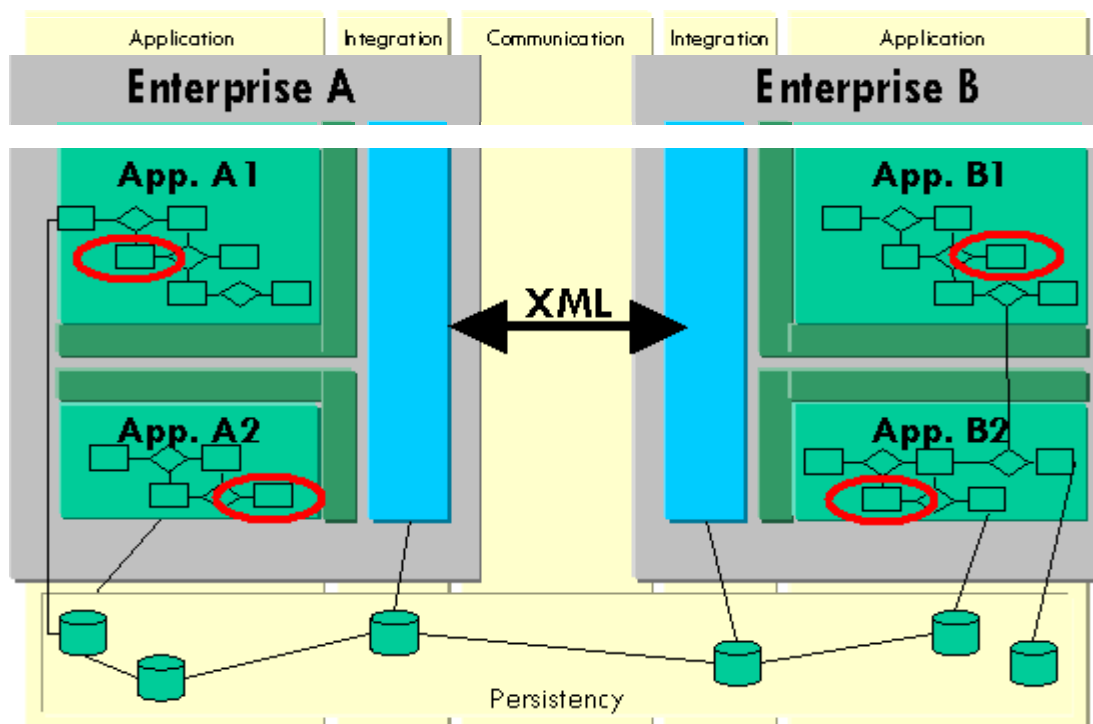


Abbildung 1: Positionierung von JAXFront®



Die in diesem Konzept behandelte Thematik positioniert sich innerhalb der Applikationsschicht. Dabei spielt es keine Rolle woher die XML Daten stammen, wie diese in die Applikation einfließen und für welche Zwecke sie genutzt werden.

1.3 Daten und ihre Repräsentation

Ist die grundlegende Struktur der Daten und die darauf operierenden Funktionen entwickelt, muss eine entsprechende Benutzeroberfläche erstellt werden. Für die Erstellung der Benutzeroberfläche stehen in den meisten Fällen weit fortgeschrittene Werkzeuge zur Verfügung. Diese bieten zwar eine gute Unterstützung um layoutbezogene Aspekte zu realisieren, die Programmierung der Verbindung zwischen Benutzerschnittstelle und den eigentlichen Daten wird jedoch vollständig dem Entwickler überlassen. Der Aufwand für die Entwicklung und Wartung solcher Bedienoberflächen ist daher sehr gross. Ändern sich die Datenstrukturen häufig, so wird das Problem der ständigen Synchronisation der Daten und deren Repräsentation noch weiter verschärft.

Durch die zu enge Koppelung zwischen Präsentations- und Applikationslogik muss bei jeder Datenstrukturänderung, die Applikations- sowie die Präsentationslogik von neuem angepasst werden. Einer sauberen Trennung zwischen Layout und Logik wird bei den heute verbreiteten Systemarchitekturen im Bereich von B2B - Anwendungen meist mit HTML ein jähes Ende gesetzt.

HTML ist eine Mischung aus Formatierung, Beschreibung und programmierter Logik. Die zusätzliche Verteilung der Programmteile auf mehrere Seiten machen eine Erweiterung bzw. eine Wiederverwendung dieser unmöglich. Änderungen in den Datenstrukturen tangieren daher oft mehrere Teile innerhalb der Systemarchitektur und sind daher mit hohem Anpassungsaufwand verbunden.

1.4 Das Bearbeiten von XML-Dokumenten

Heute gibt es eine Vielzahl von frei erhältlichen XML Parsern und Transformationsprozessoren die alle gängigen Programmiersprachen unterstützen wobei sich die ersten kommerziellen sowie frei erhältlichen „Publishing Frameworks“, Integrationsserver und Datenbanken bereits im „Krieg der XML Technologien“ bereits bewährt haben. Jedoch gibt es erst sehr wenige Werkzeuge die einen dynamischen Mechanismus anbieten um die Nachbearbeitung eines XML Dokumentes sinnvoll zu unterstützen. Gerade für die Navigation in komplexen Datenstrukturen ist eine optimierte Darstellung von grosser Wichtigkeit.

Unzählige XML Editoren erleichtern zwar dem Softwareentwickler das Arbeiten mit XML Dokumenten und bieten weitreichende Unterstützung bei der Anwendung neuer XML Technologien an, doch handelt es sich bei diesen Editoren oft um riesige Softwaremonolithen wobei die wenigsten eine saubere Programmierschnittstelle (API) anbieten. Daher sind sie für eine Verwendung innerhalb einer bestehenden Applikation meist ungeeignet. Zudem wurden sie für Entwickler geschrieben und werden auch ausschliesslich von Leuten mit fundiertem Informatikwissen verwendet. Für das Nachbearbeiten von eingehenden XML Meldungen im B2B Bereich oder für das Konfigurieren einer neu erworbenen Software werden jedoch meist keine Entwickler eingesetzt sondern Leute mit Fachkenntnissen von dem jeweiligen Sachgebiet.

Alle Editoren benutzen zur Navigation innerhalb eines XML Dokumentes eine graphische Visualisierungskomponente die die Strukturen in der Form eines Baumes darstellt. Jedes Element und dessen Attribute werden als Knoten dargestellt, und können dann direkt im Baum editiert werden. Bei extrem verschachtelten Strukturen wird diese Art der Bearbeitung jedoch viel zu unübersichtlich und zu kompliziert. Ein wesentlicher Teil eines intelligenten generischen Rendering Mechanismus ist also die Entwicklung und Verwendung geeigneter Techniken um genau diese Komplexität zu reduzieren

1.5 Erstellen von Benutzeroberflächen

Grundsätzlich gibt es zwei Möglichkeiten Benutzeroberflächen aufzubereiten.

- Statische Aufbereitung
- Dynamische Aufbereitung

Während bei einer statischen Aufbereitung das Layout und die interaktiven Regeln eng im Zusammenhang mit den darzustellenden Daten stehen, entsteht die Darstellung und die dazugehörigen Interaktionskomponenten bei der dynamischen Aufbereitung anhand von Metainformationen.



Unter dem generischen Ansatz wird im Kontext dieses Dokumentes eine dynamische Aufbereitung des Layouts ohne Fremdeinfluss aufgrund von allgemeingültigen Regeln verstanden.

Der nächste Abschnitt gibt Aufschluss über die Unterschiede zwischen statischer und dynamischer Aufbereitung sowie Probleme bei der Erstellung von statischen Benutzeroberflächen.

1.5.1 Statische vs. dynamische Erzeugung

Die Erstellung von Benutzeroberflächen nimmt in den meisten Softwareentwicklungsprozessen einen hohen Stellenwert ein. Vom eigentlichen XML Dokument bis zur sinnvollen Darstellung der beschriebenen Daten und somit einer legitimen Nachbearbeitung ist es ein langer Weg. Oft leidet die Analyse von Geschäftsprozessen oder die Definition der eigentlichen Applikationslogik unter den Limitationen der gewählten Präsentationstechnik.

In den meisten Systemarchitekturen ist das Erstellen von Layouts, die Beschreibung von Validierungsregeln und die Kopplung von Graphischen Elementen an die zu repräsentierenden Daten ein fest verdrahteter Prozess. Diese Definitionen erfolgen statisch zur Kompilationszeit.

Unter dynamischer Erzeugung einer Benutzeroberfläche wird das Erstellen des Layouts zur Laufzeit verstanden. Die meisten web-basierten Anwendungen erzeugen zwar dynamisches HTML, tun dies jedoch aufgrund von ausprogrammierten, statischen Templates. Eine weitere Möglichkeit bieten so genannte Stylesheets, um ein Ausgabeformat dynamisch zu erzeugen. Solche Templates sind meist eine Mischung aus Skripten und Platzhaltern, welche zur Laufzeit gefüllt werden. Dynamisch verhält sich in diesem Sinne nur der Inhalt der gelieferten Daten. Die beschriebenen Layoutstrukturen und deren Abhängigkeiten bleiben jedoch gleich, also statisch. Ändert sich die Struktur der zu visualisierenden Daten, so müssen diese Stylesheets angepasst werden, da sie zur Kompilationszeit Annahmen über den Strukturaufbau der darzustellenden Daten gefolgt sind.

1.5.2 Verwendung von XSLT

XSLT ist eine XML Sprache die eine Übersetzung von beliebigen XML Dokumenten in einen allgemeinen Unicode-Stream erlaubt. Die häufigste Anwendung dürfte aber die Transformation zwischen verschiedenen XML Strukturen sein. Für das Erstellen von anderen Ausgabeformaten, die mit Markierungssprachen (engl. markup languages) arbeiten, zum Beispiel HTML, wäre der Einsatz von XSLT auch denkbar. So ist das Erstellen eines einfachen HTML Outputs aufgrund eines XML Dokuments kein Problem. Was geschieht jedoch mit den XML Metadaten zu einem XML Dokument, wie etwa denen aus einem XML Schema?

Obwohl ein XML Schema auch in XML geschrieben ist und somit ein XML Dokument verkörpert, ist das Arbeiten mit Metadaten um einiges komplexer als die einfache Transformation eines in XML beschriebenen Dateninhalts in ein HTML Format. Die in einem XML Schema vorzufindenden Inhalte geben Auskunft über den logischen Aufbau und die Struktur eines XML Dokuments. Objektorientierte Ansätze wie die der Vererbung oder der Kapselung müssten also auch von XSLT (-Stylesheets) verwertet werden.

Ein weiteres Problem ist die Datenbindung. Es stellt sich die Frage, wie die in XML beschriebenen Dateninhalte mit denen innerhalb eines HTML Formulars existierenden Komponenten (Eingabefeld, List, Entscheidungsfeld etc.) verknüpft werden können. Datenänderungen müssten dann immer wieder im visualisierten XML Dokument abgebildet werden. HTML bietet hierzu keine Lösung. Zudem sollte die Möglichkeit bestehen Eingabewerte jederzeit gegen die im XML Schema beschriebenen Regeln (Standardwerte, erlaube Wertebereiche, reguläre Ausdrücke etc.) zu validieren. Einzig und allein Java-Script wäre in der Lage ein solches Vorhaben zu bewältigen. Das in XSL geschriebene Stylesheet müsste diese Validierungsskripte, aufgrund der im XML Schema beschriebenen Regeln, generieren können.

Die Idee eine Transformationssprache wie XSLT zu verwenden um eine Benutzeroberfläche aufgrund eines XML Schemas zu generieren, ist viel zu umständlich und kompliziert.

1.6 Dynamische GUI Generierung

Durch die dynamische Generierung graphischer Benutzeroberflächen auf der Basis von XML Schema Definitionen wird eine spürbare Verkürzung der Entwicklungszyklen und eine Entkopplung zwischen der Modellierung von Geschäftsmodellen und der eigentlichen Anwendungsentwicklung erreicht. Dies hat zur Folge, dass Änderungen im Geschäftsmodell (XML Schema) direkt von der Präsentationslogik nachvollzogen werden, ohne dass diese zu einer applikatorischen Änderung führt, was eine nachhaltige Stärkung der Businessseite gegenüber der IT-Fachabteilungen bedeutet.

Die Generierung von Benutzeroberflächen verlangt nach einer Analyse und Interpretation bestehender Beschreibungen in einer standardisierten Modellierungssprache. Diese Art der Darstellungsaufbereitung grenzt sich klar gegenüber heute bekannten und verwendeten Techniken zum automatischen Generieren von Benutzeroberflächen (z.B. dynamicHTML) ab. Bei den meisten Präsentationstechniken stehen layoutspezifische Eigenschaften im Vordergrund. Meist wird dabei eine graphische Benutzeroberfläche für eine bestimmte Art von Dokumenten oder eine bestimmte Datenstruktur erstellt. Die Verbindung der aktuellen Daten mit einer Visualisierungskomponente ist dann oft ein statisch ausprogrammiertes Stück Code oder eine spezielle Definitionsbeschreibung.

Die Mächtigkeit der Modellierungssprache XML Schema sollte also auch für die generische Erstellung von Benutzeroberflächen genutzt werden können. Die so erzeugten Oberflächen bieten die volle Funktionalität, die aus der Strukturbeschreibung der Daten ableitbar ist und eignen sich ausgezeichnet zur Aufbereitung von graphischen Benutzeroberflächen. Durch eine geschickte Gruppierung von Elementen mit ähnlichen Eigenschaften, einer tabellarischen Darstellung von Kollektionen und einer Navigationskomponente, welche die Baumkomplexität reduziert und somit die Navigation vereinfacht, wird eine komfortable Nachbearbeitung von XML Daten auch bei generierten Benutzeroberflächen möglich.

2 Architektur

2.1 Modell basierte Architektur

JAXFront® verwendet eine deklarative Modellierungssprache um aus einem Businessmodell direkt eine graphische Benutzeroberfläche zu generieren.

Als Modellierungssprache wird XML Schema verwendet. Dies ist ein weltweit anerkannter Standard (geführt vom W3C) für die Beschreibung von XML Datenstrukturen (Modellen). Ein XML Schema formuliert die syntaktischen und strukturellen Anforderungen eines XML Dokuments.

JAXFront® analysiert zur Laufzeit das Business Modell (XML Schema) und erzeugt (Directly Execute) aufgrund dessen eine graphische Benutzeroberfläche (Dynamically GUI). Wenn sich das Business Modell ändert, so wird auch das graphische Frontend automatisch die Änderungen übernehmen. Ein erneuter Programmieraufwand wird somit hinfällig.



Abbildung 2: Modell basierte Architektur

2.2 Konzeptionelles Modell

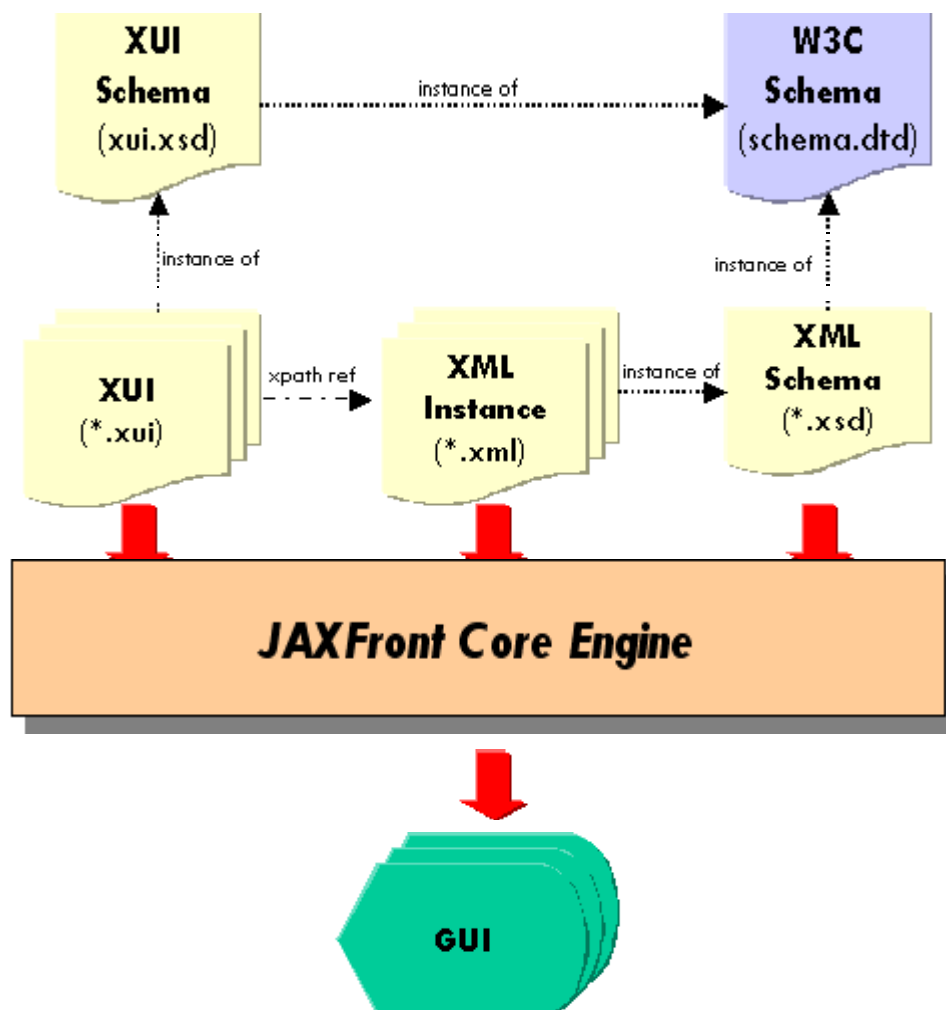


Abbildung 3: JAXFront® - Konzeptionelles Modell

Das zentrale Element ist ein XML Schema. Das Aussehen der graphischen Benutzeroberfläche wird primär durch die im XML Schema definierten Strukturen, Strukturrelationen und Kardinalitäten bestimmt. Um verschiedene Darstellungen auf denselben Strukturinformationen zu gewährleisten, kann pro gewünschte Sicht eine XML GUI Spezifikation erstellt werden. Die in der XUI Spezifikation definierten Eigenschaften beziehen sich allerdings immer auf – in einer XML Instanz vorhandene - Elemente.



Wie bei den XUI Spezifikationen können beliebig viele XML Instanzen eines XML Schemas vorliegen. Solange sie sich an die im Schema definierten Regeln halten, können diese von JAXFront® verarbeitet werden.

Das Erzeugen einer Benutzeroberfläche mittels JAXFront® besteht aus folgenden sechs Hauptkomponenten:

- **XML Schema**
Beschreibung der in einer XML Instanz gültigen Strukturrelationen, Kardinalitäten, Wertebereiche und Standardwerte.
- **XML Instance**
Beinhaltet die Daten für die Nachbearbeitung.
- **XUI**
Spezifiziert das Aussehen, Verhalten und die Ereignissteuerung der visuellen Komponenten.
- **View**
Graphische Repräsentation der zur Nachbearbeitung bereitgestellten Daten.
- **XUI Schema**
Definiert die Syntax für die GUI Beschreibungssprache „XUI“ in der Form eines W3C Schema.
- **W3C Schema**
Definiert die Syntax eines XML Schemas in Form einer DTD (Data Type Definition).
Entspricht der Menge aller XML Schemas.

2.3 Business Modell – GUI Modell

Das Businessmodell besteht aus einem standardisierten Geschäftsmodell (XML Schema) sowie einer konkreten Ausprägung (XML Instanz). Das XML Schema beschreibt die syntaktischen Anforderungen ans Geschäftsmodell, während die XML Instanz eine Konkretisierung des beschriebenen Modells darstellt. JAXFront® analysiert die Geschäftsdatenstrukturen aus dem XML Schema und erstellt eine allgemeingültige (generische) graphische Benutzeroberfläche zur Laufzeit.

Mit dem GUI-Modell können individuelle Anpassungen (unabhängig vom endgültigen Ausgabeformat) an der generierten graphischen Benutzeroberfläche vorgenommen werden. Die Präsentationslogik ist in layout- und verhaltensspezifische (behaviour) Bereiche unterteilt. Das Layoutmodell bestimmt Aussehen und Anordnung der graphischen Elemente, wobei das Behaviourmodell Regeln und Bedingungen an die graphische Benutzeroberfläche beschreibt. Die Zuordnung der Beschreibung des GUI-Modells zu den Elementen des Datenmodells geschieht mittels XPath der dem W3C-Standard entspricht.

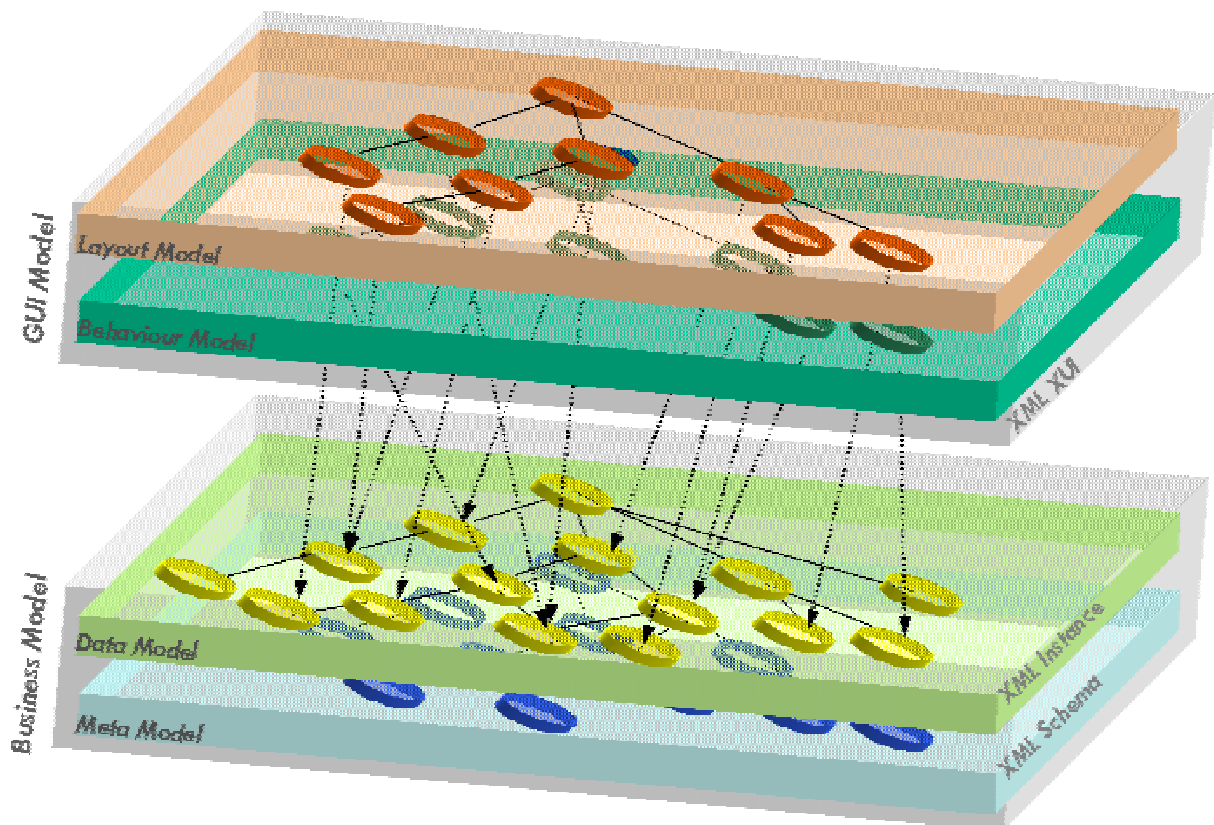


Abbildung 4: Business Modell - GUI Modell

2.4 Rollenverteilung

Die Vielfältigkeit der erzeugten Oberflächen hängt von der Anzahl und der Komplexität der XUI Spezifikationen ab. Durch das Personalisieren von Benutzeroberflächen pro Benutzergruppe können verschiedene, den jeweiligen Anforderungen angepasste, Oberflächen definiert werden. Entscheidend ist dabei, dass es sich immer um dieselbe Schemadefinition handelt.

Der Systemdesigner ist zusammen mit den Entwicklern für die Erstellung resp. Pflege des Businessmodells (XML Schema) sowie der verschiedenen GUI Spezifikationen (XUI) zuständig. Die Endbenutzer (User 1-4) arbeiten nur mit den XML Instanzen. Pro Benutzergruppe (User 1-4) können dann spezifische XUI Oberflächen definiert werden, welche nur das zulassen was nach den jeweiligen Benutzergruppenrichtlinien erlaubt ist (Rollenkonzept).

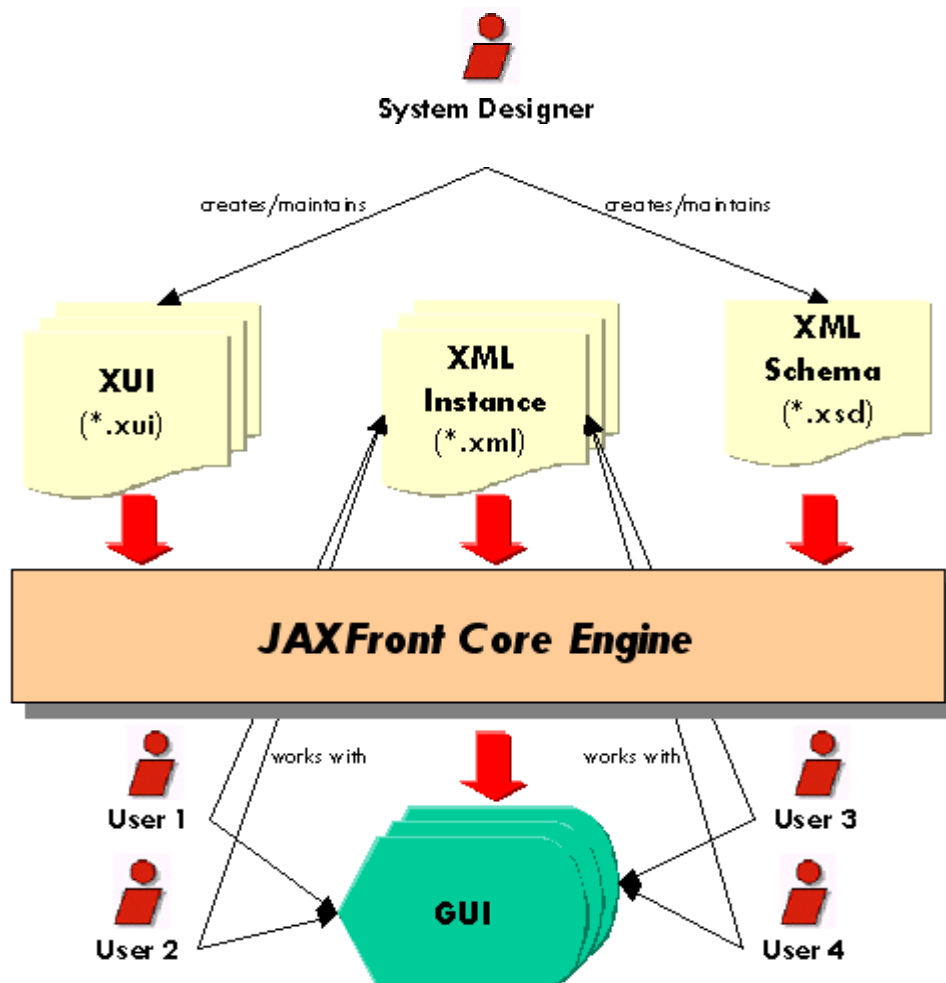


Abbildung 5: JAXFront® – Rollenkonzept

3 Core Rendering Engine

3.1 Überblick

Die JAXFront® CoreEngine ist eine 100% Java Implementation und verlangt eine lauffähige Java Virtuelle Maschine (JRE – Java Runtime Environment) Version 1.4 und höher. Die Engine besteht im Wesentlichen aus vier Hauptkomponenten.

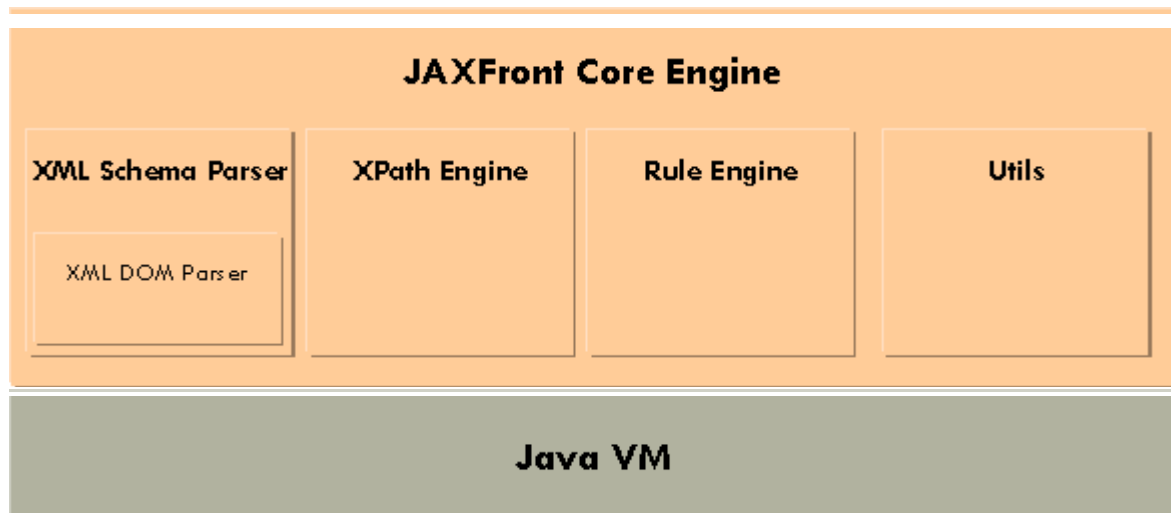


Abbildung 6: Überblick der Core-Engine

XML Schema Parser

Um eine XML Instanz gegen die definierten Bedingungen im XML Schema zu prüfen wird ein XML SchemaParser, welche die gesamte W3C Spezifikation unterstützt, benötigt. Das Erzeugen eines Document Object Models (DOM) und die XML Syntaxprüfung übernimmt der DOM Parser.

XPath Engine

Für das Ausführen von XPath Ausdrücken wird eine XPath Engine benötigt. Die XPath Engine erlaubt das Adressieren von XML Knoten aus der XML Instanz. JAXFront® verwendet eine eigene Implementation der XPath Engine Jaxen.

Rule Engine

Eine Regelmaschine unterstützt das Ausführen von einfachen oder komplexen (verschachtelten) Bedingungen. JAXFront® definiert eine Regel über ein ECA (Event-Condition-Action) – Konstrukt.

Utils

Werkzeuge für die Unterstützung von diversen applikatorischen Aufgaben, wie z.B. Logging, BeanProperties, Bildverarbeitung, etc.

3.2 Die 6 Prozessschritte

Die Core Rendering Engine ist das Herzstück von JAXFront®. Sie analysiert und interpretiert die Strukturinformationen aus dem XML Schema, verarbeitet die spezifischen Layoutangaben und bindet eine existierende XML Instanz an die graphische Benutzeroberfläche. Bei der Visualisierung von JAXFront® Benutzeroberflächen werden die in Abbildung 7 dargestellten Prozesse durchgeführt.

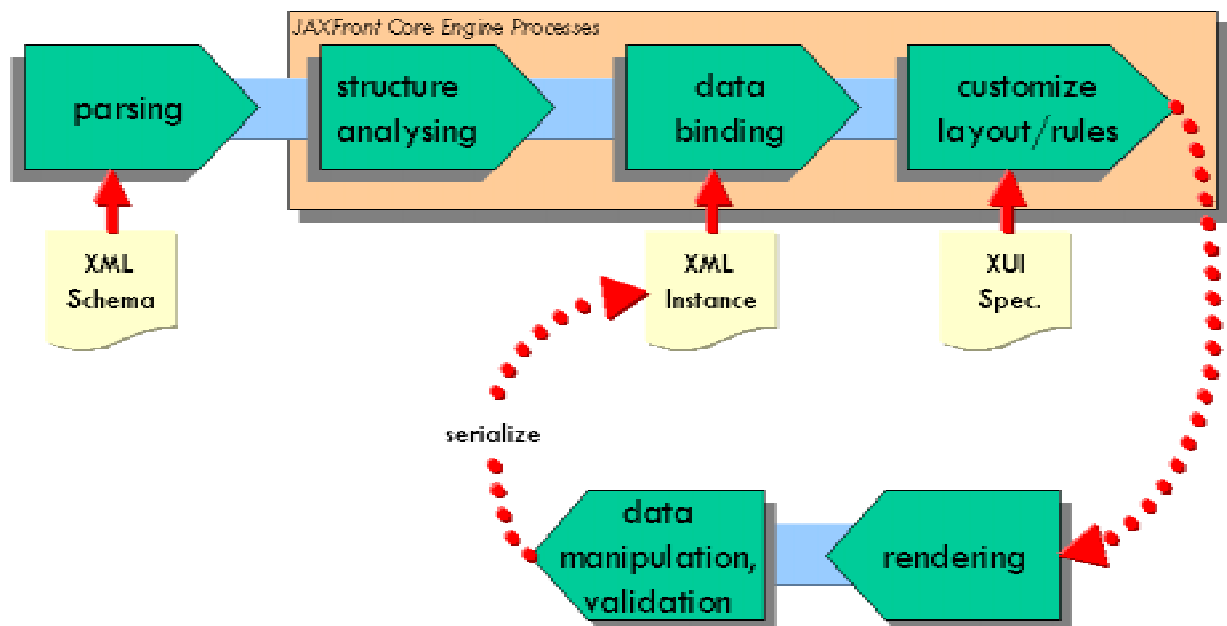


Abbildung 7: JAXFront® - Core Rendering Engine

❶ parsing

Unter Parsing wird die allgemeine Syntexanalyse eines Textes verstanden. Hier wird das XML Dokument auf seine Syntax hin geprüft und für die Weiterverarbeitung bereitgestellt.

❷ structure analysing

Dieser Prozess analysiert ein XML Schema und erstellt aufgrund von allgemeingültigen Darstellungsregeln die Layoutstruktur.

❸ data binding

Bindung der in der XML Instanz enthaltenen Daten zur Laufzeit an die visuellen Komponenten.

❹ customize layout/rules

Anpassung der Benutzeroberfläche an die jeweiligen Bedürfnisse.

❺ rendering

Wiedergabe der Layoutinformationen auf einer graphischen Benutzeroberfläche.

❻ data manipulation/validation

Die Datenmanipulation sowie die Validierung erfolgen nun interaktiv mit der erzeugten graphischen Benutzeroberfläche. Am Ende wird das Dokument gespeichert und wieder als XML Instanz serialisiert.

Zuerst wird das XML Schema auf die Syntax (parsing) sowie auf die Gültigkeit (entspricht das Schema einem W3C gültigem XMLSchema) geprüft. Falls das Dokument einem vom W3C definierten XML Schema entspricht, wird nun die Struktur analysiert (apply generic rendering rules). Das Resultat aus diesen zwei Prozessschritten ist ein typisiertes JAXFront® DOM (genauere Informationen finden Sie im Kapitel 3.4).

Ist eine GUI Beschreibung (XUI) vorhanden, wird die zu erzeugende graphische Benutzeroberfläche daraufhin angepasst (personalize GUI). Beim „data binding“ werden die Instanzdaten aus einem XML Dokument an das JAXFront® DOM gebunden. Falls keine XMLInstanz vorhanden ist, wird eine „dummy“ Instanz erzeugt (create XML Instance).

Der letzte Schritt ist die Ausgabe (rendering) auf einen bestimmten GUI Kanal. Die graphische Benutzeroberfläche wird mit den Möglichkeiten des gewählten Toolkits (JavaSwing, HTML, PDF,...) erzeugt. Die Manipulation der darunterliegenden XML Daten kann nun interaktiv über die erzeugte Benutzeroberfläche erfolgen (apply changes).

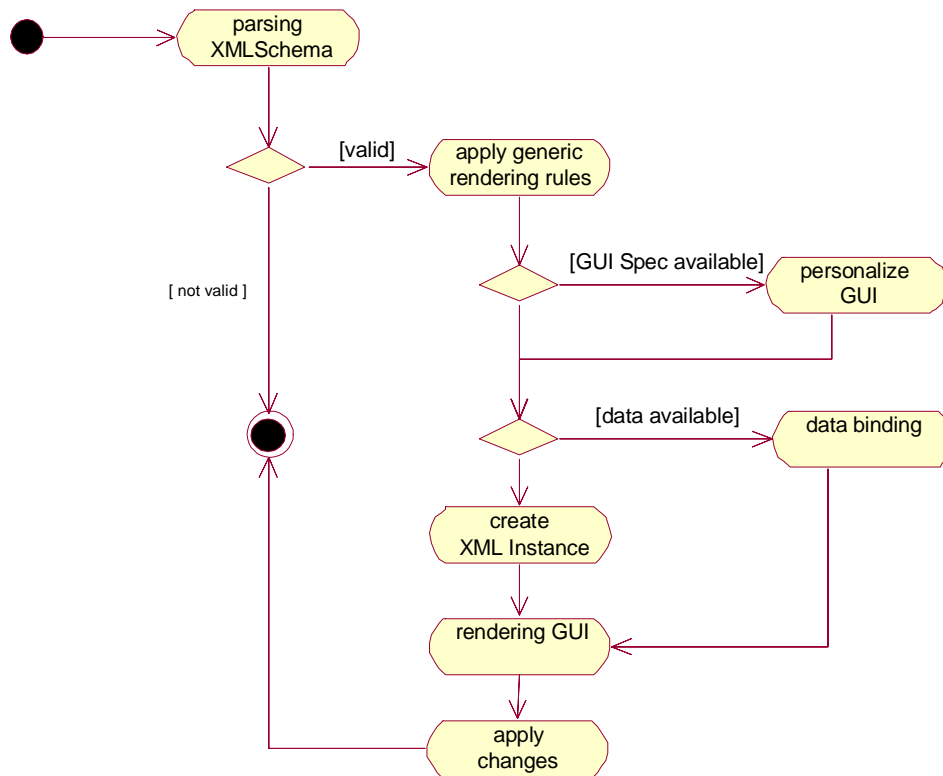
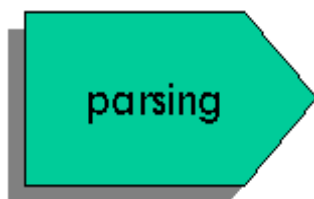


Abbildung 8: Ablaufdiagramm der Core Rendering Engine

3.3 Parsing



Der erste Schritt in der Verarbeitung eines XML Dokuments ist die Syntexanalyse und eine eventuelle Validierung gegen ein DTD oder XML Schema. Unter „Parsing“ verstehen wir in diesem Kontext jedoch mehr als eine reine Syntexanalyse oder eine initiale Validierung gegen ein DTD oder Schema. Es gilt das XML Dokument zuerst auf seine Syntax zu prüfen und es dann in einer geeigneten Form für die

Weiterverarbeitung in einer Programmiersprache bereitzustellen.

Das W3C kennt zwei verschiedene standardisierte Konzepte für dieses Vorgehen. Zum Parsen eines XML-Dokuments wird entweder ein DOM-Parser, der eine Baumstruktur aufbaut oder ein SAX-Parser verwendet, der bei bestimmten Struktureigenschaften des Dokuments entsprechende Ereignisse auslöst. Beide Konzepte haben ihre Stärken und ihre Schwächen, auf die hier jedoch nicht weiter eingegangen wird.

Da bei einem SAX-Parser alle Struktur- und nicht verarbeitete Informationen verloren gehen, und des Weiteren SAX nur zum Lesen von XML-Dokumenten verwendet werden kann, verwenden wir in diesem Konzept ausschliesslich DOM-Parser. Mit einem DOM ist es möglich, die eingelesenen Daten zu editieren, die Strukturen zu erweitern und das ganze wieder als XML-Dokument zurückzuspeichern.

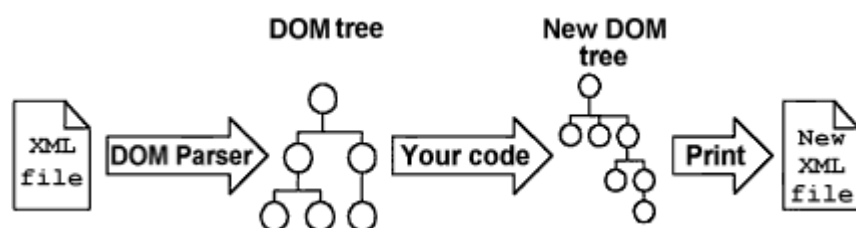


Abbildung 9: W3C DOM-Konzept

Validierend oder nicht validierend?

Aufgabe jedes Parsers, ob validierend oder nicht validierend, ist es, den zugrundeliegenden XML-Quelltext zu überprüfen und Verletzungen gegenüber der XML-Syntaxspezifikation zu erkennen.

Validierende Parser überprüfen darüber hinaus auch die Struktur eines XML-Dokuments. Unter dieser Strukturüberprüfung ist die exakte Übereinstimmung des XML-Dokuments mit der in einem DTD oder XML-Schema definierten Struktur zu verstehen.

	DTD		XML Schema	
	validierend	nicht-validierend	validierend	nicht-validierend
DOM				
SAX				

Abbildung 10: XML Parsertypen

Da für das in JAXFront® angewandte Konzept XML-Schema von zentraler Bedeutung ist, ist ein validierender Parser der die neueste XML-Schemaspezifikation unterstützt, unerlässlich. Zudem muss der Parser die Möglichkeit bieten zur Laufzeit ein vollständiges DOM gegen ein sich im Speicher befindendes XML Schema zu prüfen, und nicht nur initial während der Syntaxanalyse.

3.4 Struktur Analyse



Der folgende Abschnitt zeigt Regeln auf, um die in einem XML Schema definierte Elemente allgemeingültigen, visuellen Komponenten zuzuordnen. Diese Komponenten entsprechen genau definierten Darstellungstypen und ermöglichen somit detaillierte Aussagen über deren visuelle Repräsentation zu treffen. Nach einer eingehenden Analyse aller uns zur Verfügung stehenden graphischen Visualisierungskomponenten aus dem Java Swing – Toolkit, konnten drei grundsätzlich verschiedene Arten gefunden werden um Informationen visuell darzustellen:

Einfache, atomare: Komponenten zur Darstellung einfacher Werte (gemeint sind primitive Datentypen wie etwa boolean, string, integer etc.). Beispiele dafür sind: JTextField, JCheckBox, JLabel etc.

Gruppen: Komponenten die einen Platzhalter (Container) für zusammengehörige Werte darstellen. Beispiele sind: JBorder, JTabPane, JScrollPane etc.

Listen: Komponenten die eine Ansammlung identischer Werttypen darstellen und das Arbeiten mit einzelnen Einträgen (erstellen, ändern & löschen) erlaubt. Beispiele: JTable, JComboBox, JList etc.









Grundsätzlich kennt die XML Schemaspezifikation nur zwei Arten von Elementtypen, die einfachen (engl. simple) und die komplexen (engl. complex). Ein komplexes Element zeichnet sich dadurch aus, dass es eine Menge von Unterelementen definieren kann. Bei den einfachen Elementen spricht man von primitiven (engl. primitive) Datentypen die nur einen Wert repräsentieren können. Diese primitiven Datentypen sind sozusagen Datenbehälter und bilden, weil sie keine Unterelemente definieren können, die Menge aller Blätter (engl. leafs) innerhalb eines XML Baumes.

Für die Zuweisung eines Darstellungstypen gibt es jedoch noch mehr zu beachten als nur die Differenzierung zwischen einfachen und komplexen Typen. So ist es möglich das ein nach XML Schema definiertes, komplexes Element keineswegs komplex sein muss um es sinnvoll Darstellen zu können. Viel wichtiger ist es, den Darstellungstypen anhand der Topologie eines im XML Schema definierten Elementes zu identifizieren. So ist zum Beispiel ein im XML Schema definierter komplexer Typ keineswegs für den Darstellungsmechanismus komplex, wenn dieser nur aus primitiven Unterelementen besteht.

Unter der Topologie eines XML Schema Elements verstehen wir die Zusammensetzung der Unterelemente, dabei werden alle direkten Kind'selemente (engl. children) und deren Darstellungstypen in betracht gezogen. Entscheidend bei der Klassifizierung ist die Art und Menge der Darstellungstypen aller Unterelemente.

Durch die Unterscheidung in drei verschiedene Arten der Informationspräsentation und die Differenzierung ob eine Gruppe nur aus einfachen oder aus komplexen Elementen besteht, ergeben sich sechs weitere Typen die zur Visualisierung von XML Schemaelemente entscheidend sind. Jedem dieser Darstellungstypen ist ein Symbol zugeordnet und wird nachfolgend kurz beschrieben.

-  **SimpleType**
sind die Blätter eines Baumes (engl. leaf nodes) und definieren primitive Datenelemente oder Attribute (boolean, integer, string etc.).
-  **SimpleGroup**
beinhalten eine fest definierte Menge von einfachen Typen (SimpleTypes), einfacher Kollektionen (SimpleTypeList) oder einfachen Gruppen (SimpleGroup).
-  **ComplexGroup**
beinhaltet eine bis mehrere komplexe Gruppen oder mindestens eine einfache Gruppe bzw. Kollektionen in Kombination mit beliebig vielen einfachen Typen (SimpleType).
-  **SimpleTypeList**
ist eine Kollektion des gleichen einfachen Typ's mit einer Kardinalität grösser als eins.
-  **SimpleGroupList**
ist eine Kollektion einer einfachen Gruppe (SimpleGroup) mit einer Kardinalität grösser als eins.
-  **ComplexGroupList**
ist eine Kollektion einer komplexen Gruppe (ComplexGroup) mit einer Kardinalität grösser als eins.



Ziel ist es jedem definierten Element eines XML Schemas genau einen Darstellungstypen zuzuweisen.

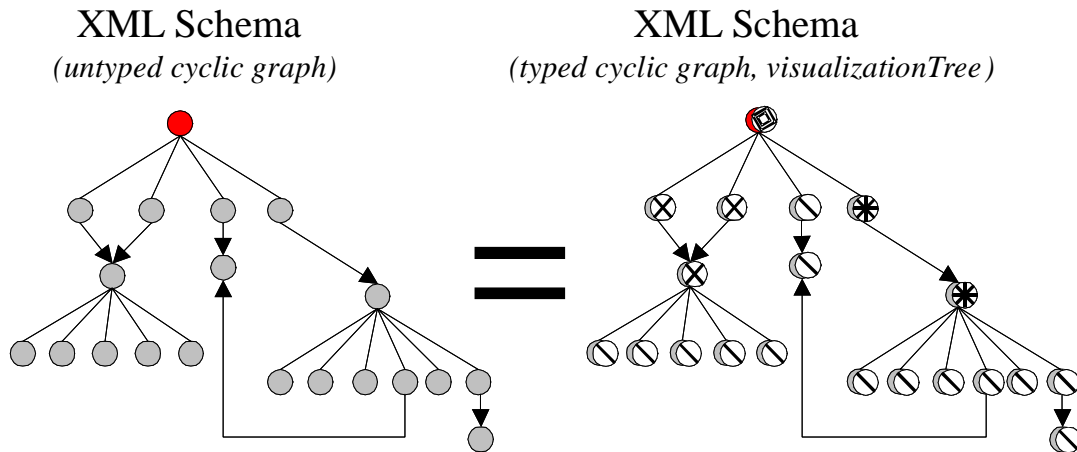


Abbildung 11: XMLSchema - Bildung eines Graphen

Das Resultat dieses Prozesses ist ein typisierter DOM – Tree, auch Visualisierungsbaum genannt. Die Anwendung der oben definierten Regeln und die Erzeugung der Darstellungstypen ist im Kapitel 0 beschrieben. Jedem Darstellungstyp ist bekannt wie er sich visuell zu repräsentieren hat.

3.5 Data binding



Unter dem Begriff data binding verstehen wir in diesem Kontext nicht das Binden von XML Datenquellen an bestehende oder neu zu erstellende Java Klassen sondern das Binden der XML Instanz Daten an die jeweiligen graphischen Elemente. Für das Arbeiten mit einem XML Dokumenteninhalte verwenden wir die Programmierschnittstelle DOM und arbeiten mit Objekten des Typs *Node*.

Dieser „binding“ Prozess bindet die XML Instanzdaten an graphische Visualisierungskomponenten (JAXFront® Typen) und ermöglicht somit Änderungen auf der Benutzeroberfläche automatisch in dem XML Instanz – Dokument nachzuführen.

Nach dem im Abschnitt 3.3 beschriebenen Vorgang der Syntaxanalyse und der Erzeugung eines Objektgraphen (DOM) müssen die einzelnen Knoten aus der XML Instanz mit einem typisierten Knoten des Visualisierungsbaumes verbunden (binding) werden.

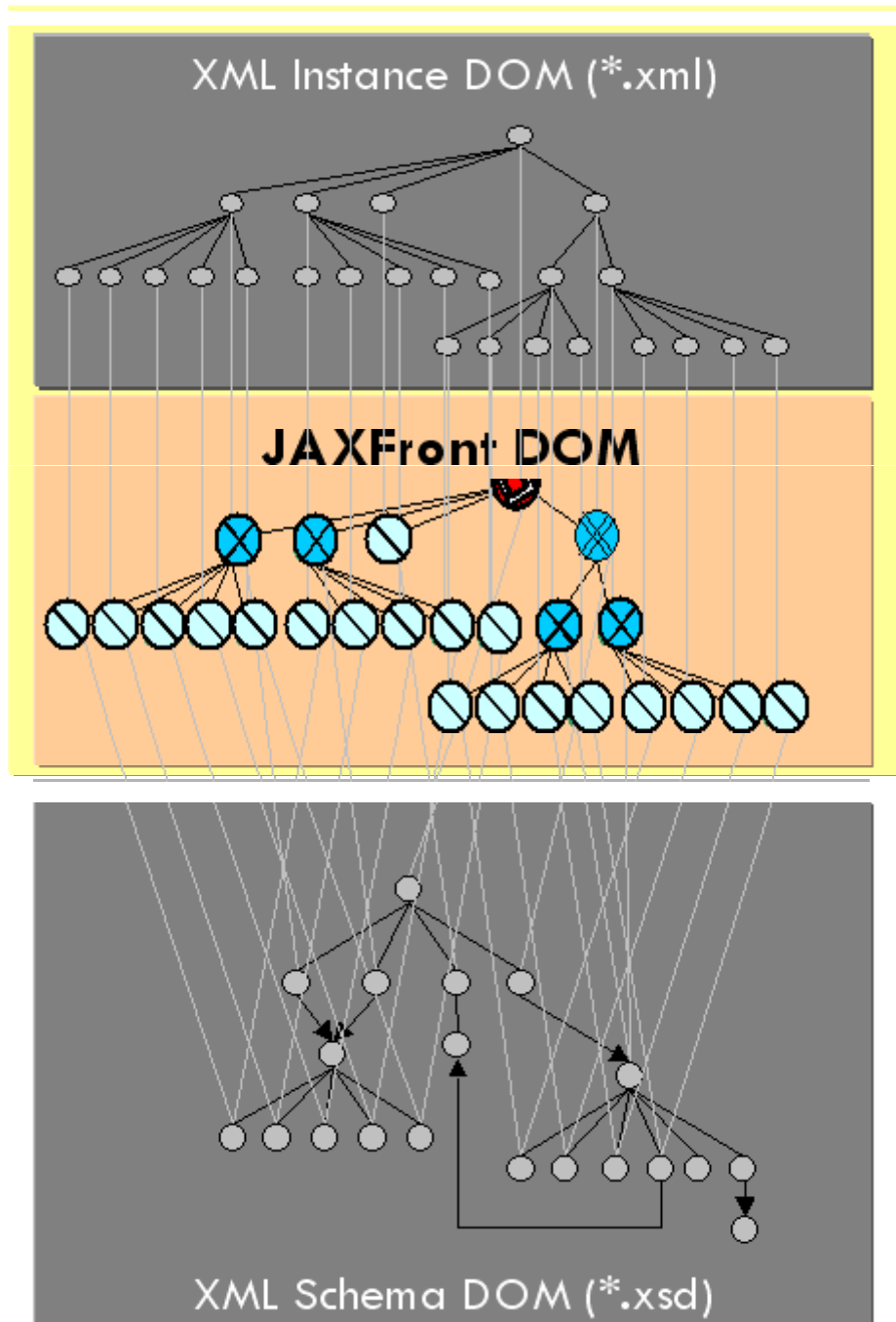


Abbildung 12: Data-Binding an ein JAXFront® DOM

Der Begriff JAXFront® DOM verkörpert ein Objektmodell in dem jedes Object einen Knoten aus Instanz DOM, XML Schema DOM und dazugehörigen Datentypen zugeordnet wurde. Für jeden Knoten aus dem XML Instanz DOM wird also ein Object kreiert, welches zwei Referenzknoten besitzt. Eine Referenz zeigt auf die eigentlichen Dateninhalte der XML Instanz, die andere verweist auf den im XML Schema definierten Datentypen und dessen Darstellungstypen.

3.6 Personalisierung der Benutzeroberfläche

**customize
layout/rules**

Um eine generisch erzeugte Benutzeroberfläche individuellen Bedürfnissen anzupassen, muss die Möglichkeit bestehen, die erzeugten Darstellungstypen deklarativ zu parametrisieren. Zu diesem Zweck muss der gesamte Visualisierungsbaum eines XML Schemas in ein benutzerfreundlicheres Format überführt werden. Dieses Format soll einfach zu verstehen und erweiterbar sein, da

die Parameter für die genauere Spezifizierung von graphischen Elementen sehr umfassend sind. Die erstellte Spezifikation soll abstrakt, das heisst unabhängig von einem bestimmten Ausgabeformat oder Programmiersprache gehalten sein. Es wäre denkbar für jedes zu erstellende Ausgabeformat eine eigene GUI Spezifikationsprache zu entwerfen. Das folgende Konzept wird sich auf das Erzeugen von Java Swing Benutzeroberflächen konzentrieren.

Für solch eine GUI Spezifizierung bietet sich XML an. XML ist lesbar, erweiterbar und kann mit einer Vielzahl von XML Editoren (auch mit der hier beschriebenen generischen Layout Rendering Engine) einfach nachbearbeitet werden. Die Definition der GUI Spezifikationsprache (eXtensible User Interface, kurz XUI) wurde mit XML Schema erstellt (xui.xsd) und definiert alle Parametrisierungsmöglichkeiten der jeweiligen Darstellungstypen.

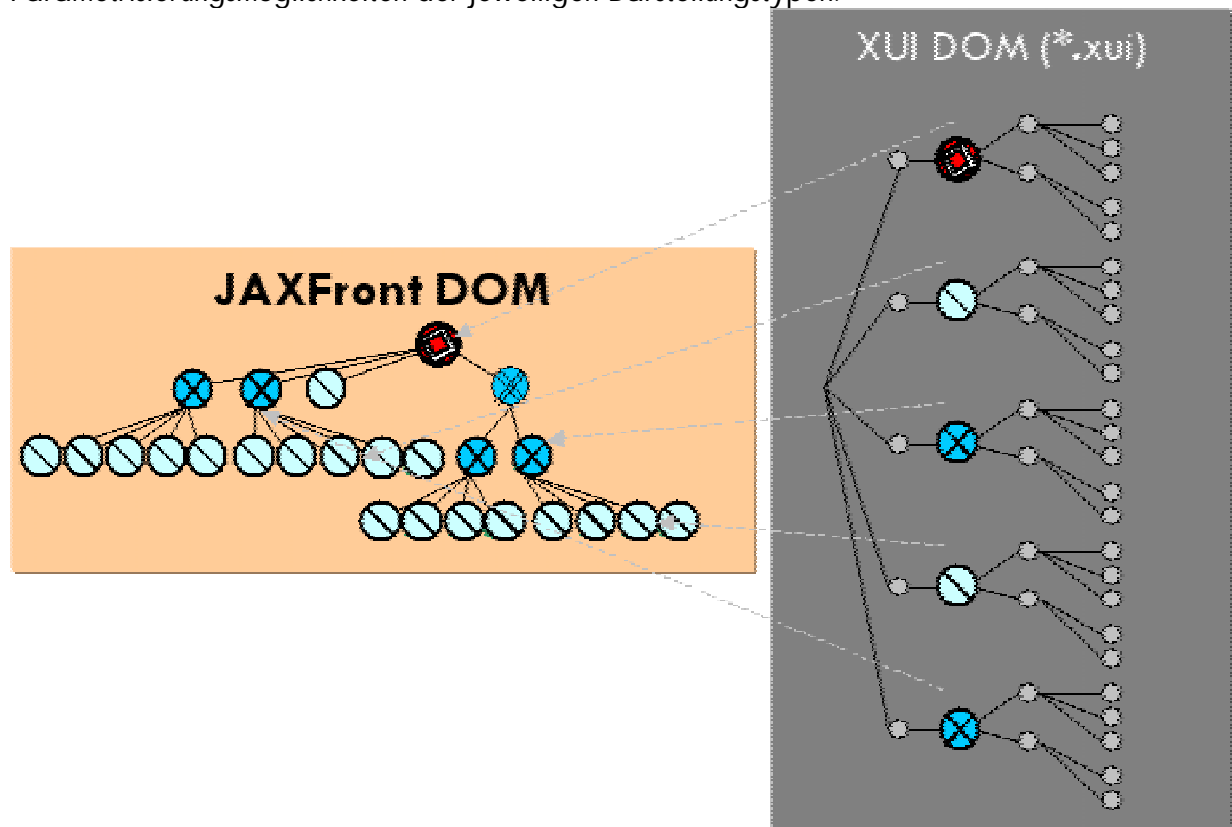
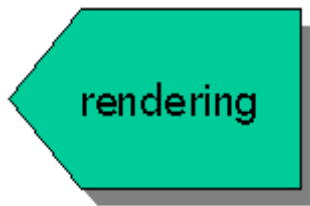


Abbildung 13: Verarbeitung der XUI-Komponenten

Jeder JAXFront® DOM Knoten kann mit einem Eintrag in der GUI – Spezifikationsdatei (XUI DOM) zusätzlich Parametrisiert werden. Die Problematik der Personalisierung – bzw. der Möglichkeit das Layout einzelner Komponenten seinen eigenen Bedürfnissen anzupassen, ist im XUI Handbuch genauer beschrieben.

3.7 Rendering



Unter dem Begriff *Rendering* wird im Allgemeinen die Wiedergabe einer dreidimensionalen Darstellung unter Berücksichtigung aller Lichtquellen unter Verwendung von verschiedenen Schattierungsverfahren verstanden. Im Kontext dieses Dokumentes verstehen wir unter Rendering das Aufbereiten und Visualisieren eines Darstellungstypen für ein bestimmtes Ausgabeformat (HTML, WML, Java Swing UI etc.). In unserem Falle sind das Java (Swing GUI) und HTML Benutzeroberflächen.

Die Visualisierung der Darstellungstypen wird in den folgenden Abschnitten anhand einer Java zentrierten Lösung aufgezeigt. Das JDK integriert die Grafikbibliothek namens *Swing* welche sich sehr gut für die Visualisierung dieser Darstellungstypen eignet, da die Architektur aller in Swing vorzufindenden graphischen Visualisierungskomponenten dem MVC (Model-View-Controller) – Paradigma folgt. Im MVC Konzept wird auch eine Trennung zwischen den eigentlichen Daten (Model) und der Visualisierung (View, Controller), wie wir sie in unserem JAXFront® DOM (siehe Kapitel 0) vorfinden, zugrunde gelegt.

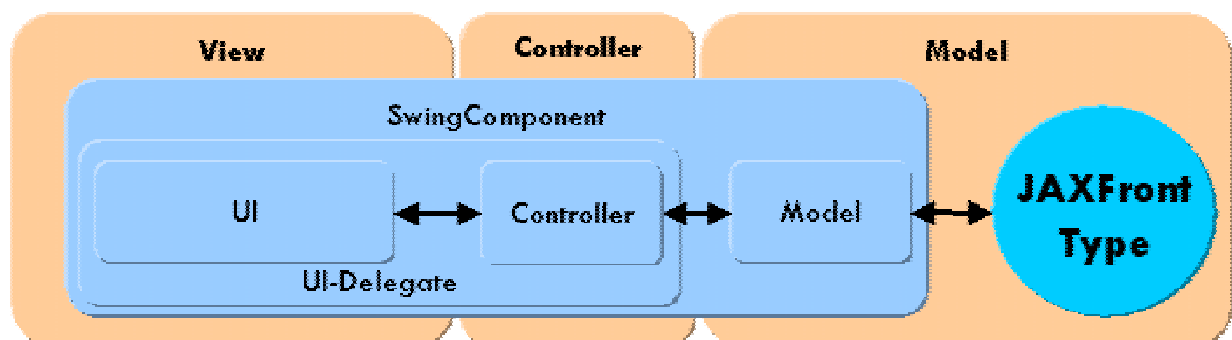


Abbildung 14: Model-View-Controller Paradigma von Java Swing

3.8 Interaktive Datenmanipulation, Validierung



Die generierte Benutzeroberfläche erlaubt es, die darunterliegenden XML Instanzdaten interaktiv zu bearbeiten. Jedem GUI-Eingabeelement liegt ein JAXFront® Type zugrunde. Beim Erfassen resp. Mutieren des Eingabewertes wird der aktuelle Wert des Feldes gegen die im XMLSchema definierten Facetten sowie der im XUI hinterlegten ECA-Regeln geprüft. Weiter werden auch Kardinalitätsbedingungen aus dem XMLSchema geprüft. Verstößt ein Wert gegen diese Validierungsregeln, so wird das betroffene Eingabefeld als fehlerhaft markiert, eine Fehlermeldung erstellt die in einer Fehlerliste aufgenommen wird.

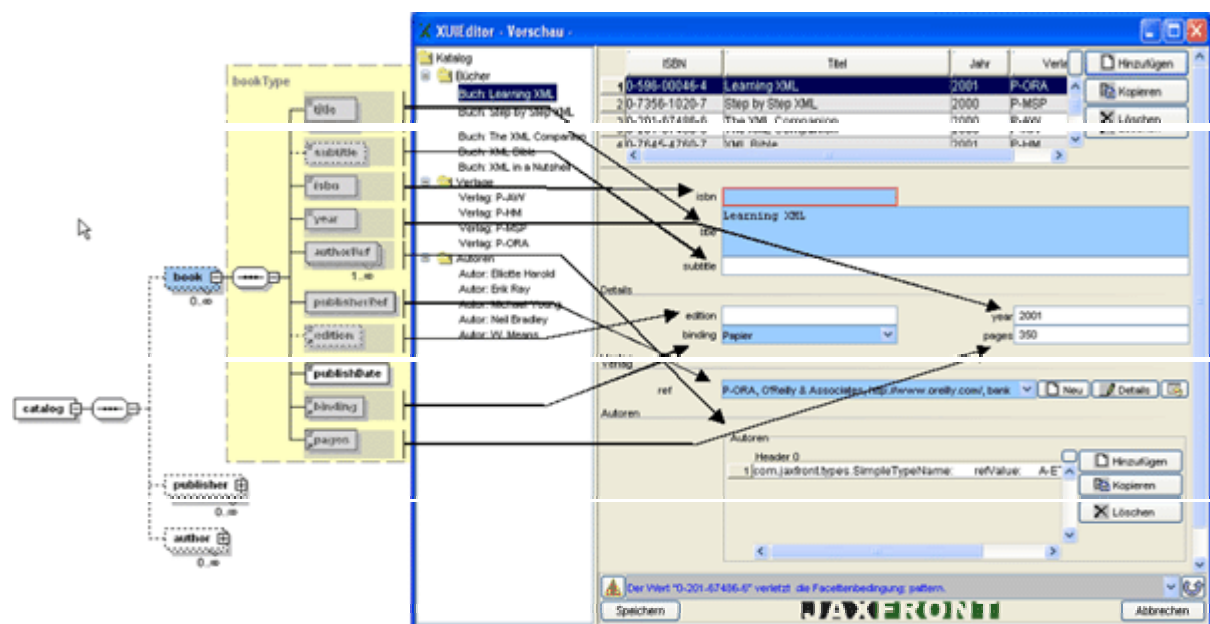


Abbildung 15: Interaktive Datenmanipulation & Validierung

4 Systemkomponenten

4.1 Überblick

Einer der wichtigsten Bestandteile der JAXFront® XML Rendering Technologie ist die Fähigkeit, die Ausgabe auf verschiedenen User Interfaces darzustellen (UI Channels). Bis jetzt existieren drei unterschiedliche Renderingverfahren (Channels), die auf dem gleichen Daten Modell arbeiten: JavaSwing, HTML und PDF. Weitere GUI - Kanäle wie zum Beispiel WML sind in Planung und Dank der Komponentenbauweise von JAXFront® in kurzer Zeit realisierbar.

Der XUIEditor ist die JAXFront® Entwicklungsumgebung zur Erstellung und Modifikation von XUI-Beschreibungen.

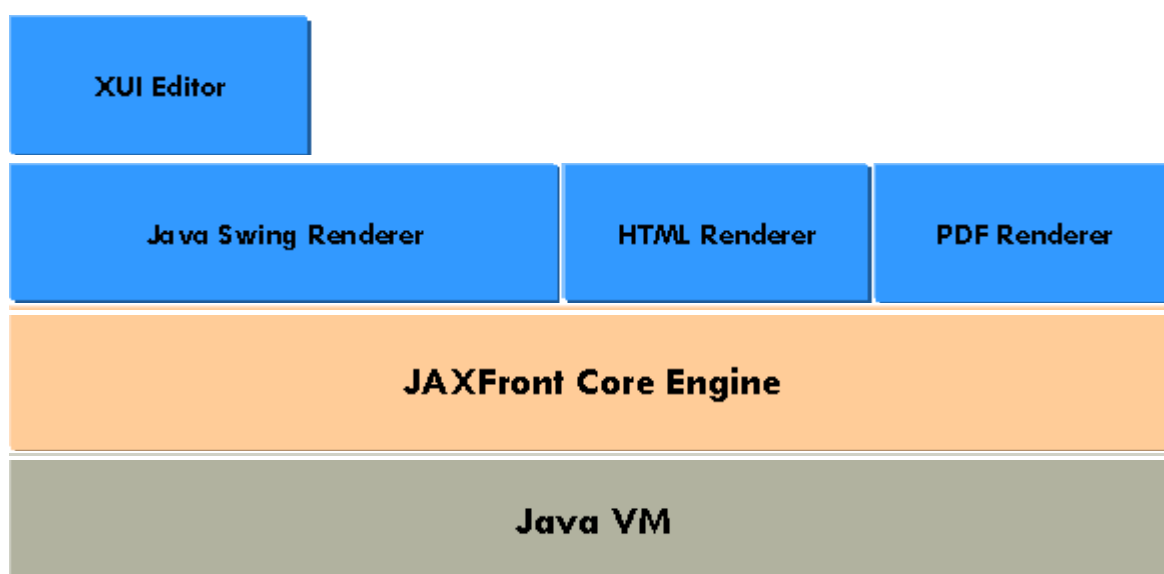


Abbildung 16: JAXFront® Systemkomponenten

SwingRenderer

Der JavaSwing Renderer generiert zur Laufzeit, dynamisch graphische Benutzeroberflächen unter Verwendung des Java Swing Graphic Toolkit's. Der Swing Graphic Toolkit ist ab der Version 1.2 Bestandteil des JDK/JRE.

HTMLRenderer

Der HTMLRenderer erzeugt HTML Seiten zur Anzeige innerhalb eines Webbrowsers.

PDFRenderer

Der PDFRenderer erstellt für ein bestehendes XUI ein eigenes PDF-XUI zum Drucken von Formularen auf Papier.

XUIEditor

Basiert auf dem Java Swing Renderer und ermöglicht das Erstellen resp. Bearbeiten von XUI-Dateien.



Fast die gesamte Benutzeroberfläche des XUIEditor wurde selbst mit der JAXFront® Rendering Engine erstellt. Dazu wurde das der XUI-Sprache zugrunde liegende XMLSchema (xui.xsd) verwendet.

4.2 Java Swing Renderer

Die Initial entstandene Benutzeroberfläche basiert auf Java Swing Komponenten und bietet übersichtliche, anwenderfreundliche und durch den Einsatz von XUI's leicht anpassbare (personalisierbare) GUI's. Hinzu kommt der Vorteil, dass Java-basierende Applikationen Plattformunabhängig und somit auf unterschiedlichen Betriebssystemen lauffähig sind. Weiter besteht die Möglichkeit eigene Plug-Ins zu erstellen, die mittels XUI an beliebigen Stellen auf der Benutzeroberfläche platziert werden können um somit eine noch individuellere Darstellung der gewünschten Informationen zu ermöglichen.

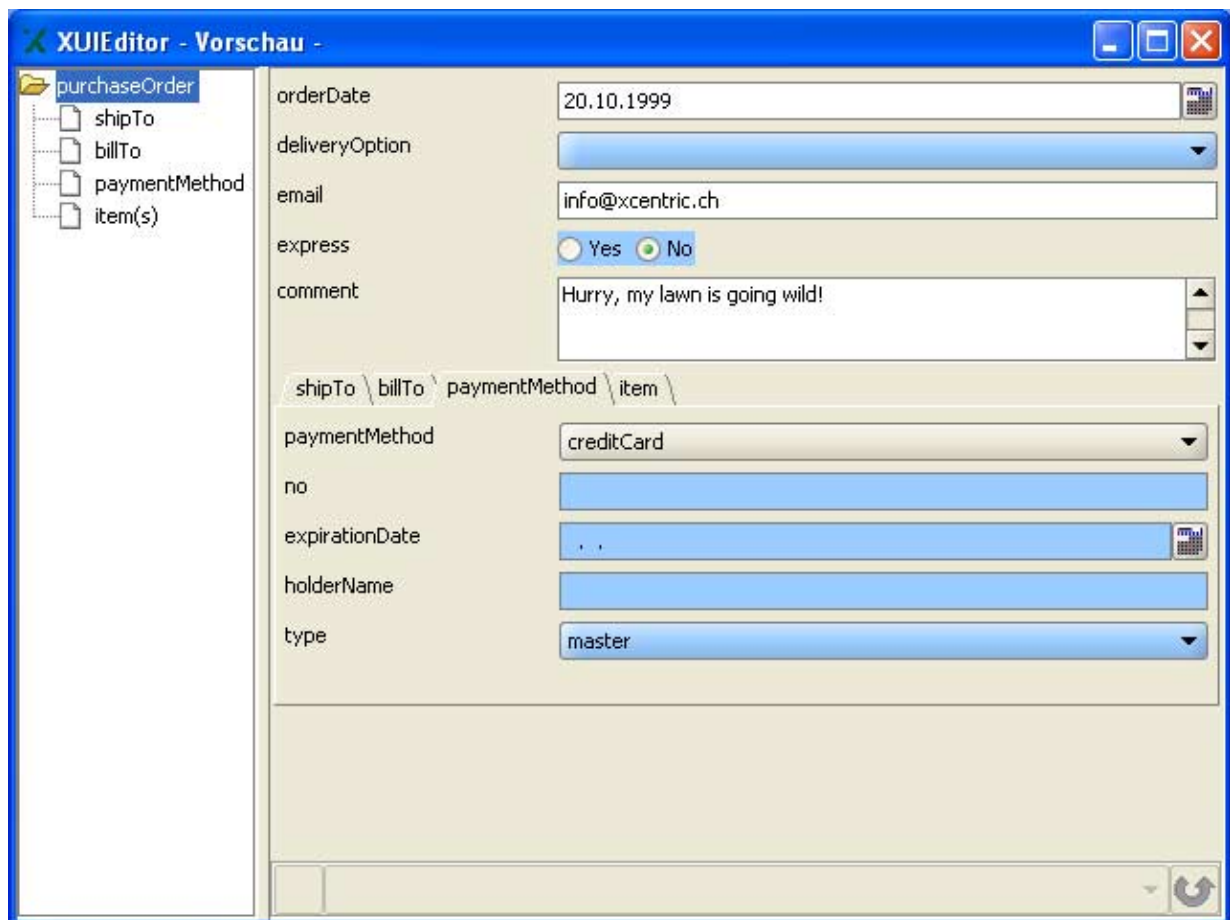


Abbildung 17: SwingRenderer (Screenshot)

4.3 HTML Renderer

Das HTML-GUI von JAXFront[®] wurde entwickelt, um der wachsenden Nachfrage unserer Kunden nach einer HTML-Oberfläche gerecht zu werden. Der technische Aufbau sowie die Bedienung der HTML-Variante von JAXFront[®] weicht technologisch bedingt in einigen Bereichen von der Java Swing Version ab. Einer der besonderen Unterschiede ist zum Beispiel, dass die HTML-Benutzeroberfläche als Thin-Client mittels Java Servlet Technologie realisiert wurde, im Gegensatz zur Java Swing-Oberfläche die eine smarte Fat-Client Lösung darstellt. Ein weiterer Punkt ist die Möglichkeit, ein selbst definiertes CSS (Cascading Style Sheet) für die grafische Darstellung der HTML-Komponenten zu erstellen, um somit mit wenigen Schritten die gewünschte Oberfläche zu erhalten. Dank des Einsatzes von JavaScript werden einfache Bedingungen und Einschränkungen bereits bei der Eingabe geprüft und somit der Datenaustausch zwischen Browser und Server erheblich verringert.

Browser

Die JAXFront[®] HTML RenderingEngine wurde für den Internet Explorer 6.0 konzipiert und es können beim Einsatz anderen Browsertypen „Refresh Probleme“ auftreten.

Servlet

Für den Betrieb der JAXFront[®] HTML RenderingEngine wird eine ServletEngine (Servlet-API 2.2+) benötigt.

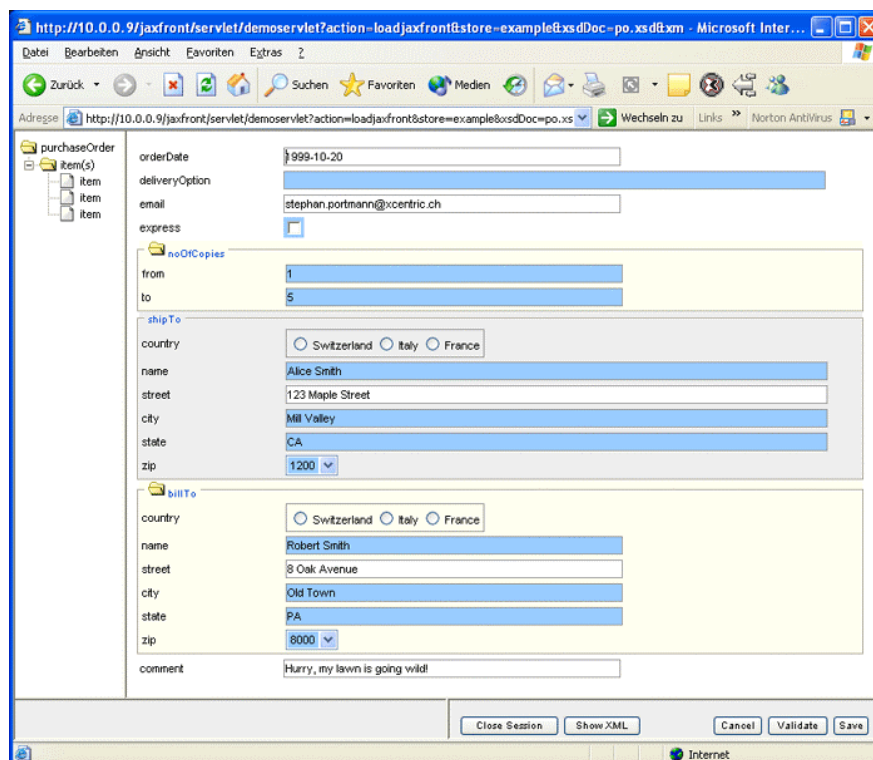


Abbildung 18: HTML Renderer (Screenshot)

4.4 PDF Renderer

Der PDF Renderer erzeugt aus einem JAXFront® DOM eine PDF Datei. Über den PDF Designer kann das Aussehen des PDF Dokuments gesteuert werden. So besteht zum Beispiel die Möglichkeit eine eigene Kopfzeile, die zu verwendenden Schriftarten und Grössen global zu setzen.

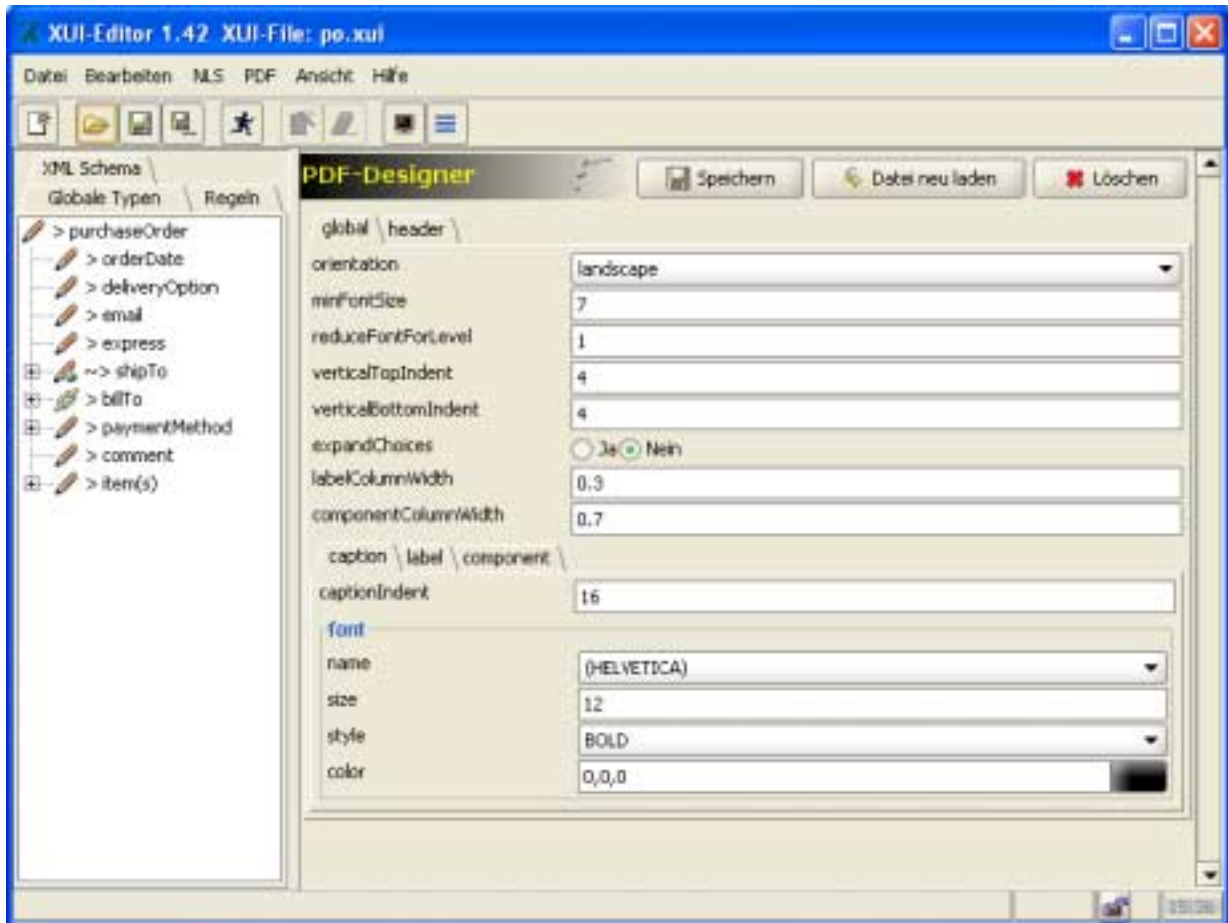


Abbildung 19: PDF Designer (Screenshot)



Jeder JAXFront® Type (einfache Typen, Gruppen und Listen) kennt seine Darstellungsregeln um sich innerhalb eines PDF Dokuments darzustellen.

Abbildung 20: Erzeugtes PDF Dokument

4.5 XUI Editor

Damit das Erstellen von XUIs zum Kinderspiel wird, wurde eigens dafür ein spezielles Tool entwickelt: der XUI Editor. Diese auf Java Swing basierende Applikation erlaubt ein schnelles und effizientes Anpassen der gewünschten Benutzeroberfläche. So ist es zum Beispiel mit nur einem Maus Klick möglich, die Darstellung eines Knoten von Baum auf Tab-Ansicht umzustellen, und diese danach von einem Renderer (Java Swing oder HTML) darzustellen.

 Das XML-Schema (zwingend), die XML-Instanz (optional) und die XUI-Definition (optional) können nach dem Start über den Menüpunkt (File/open...) ausgewählt werden:

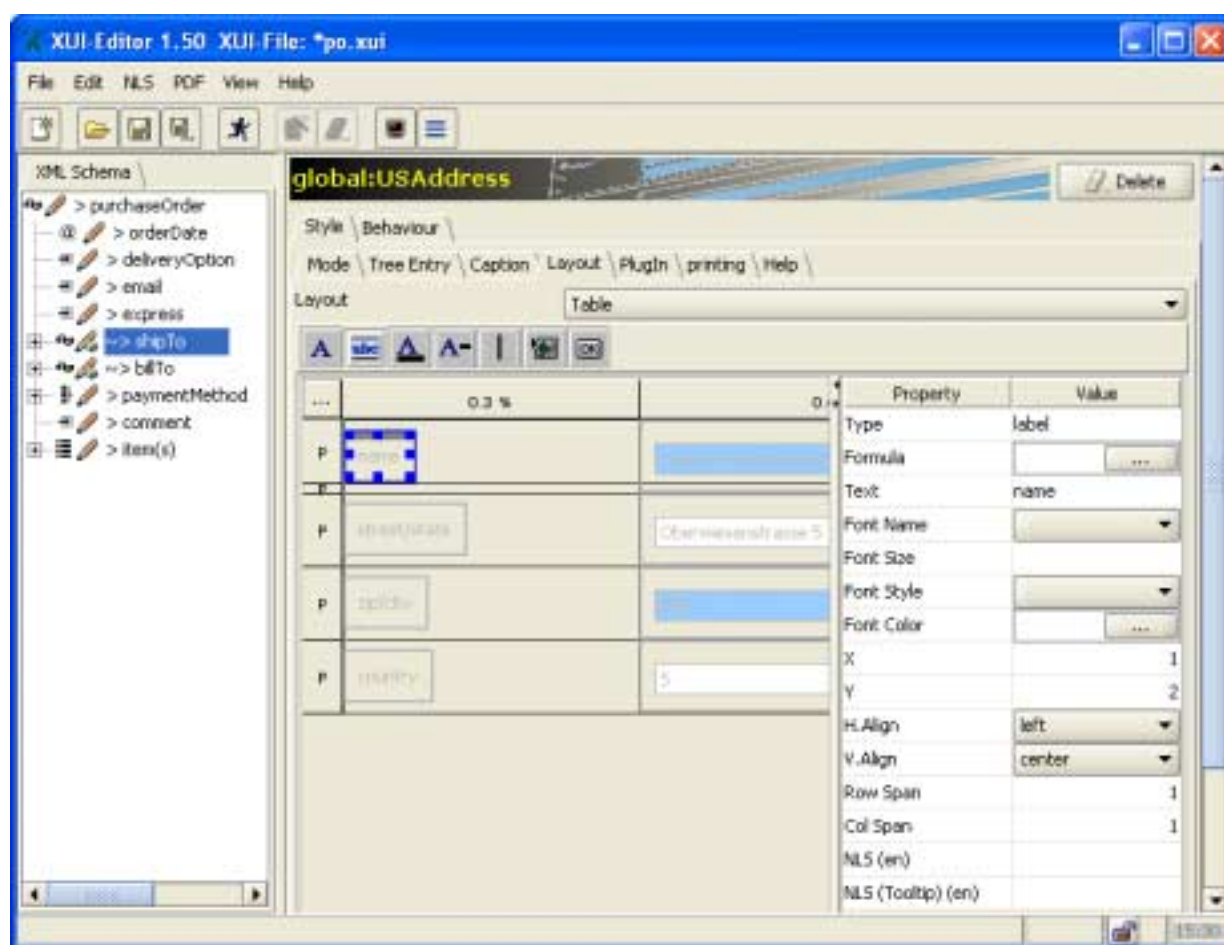


Abbildung 21: XUIEditor (Screenshot)

5 Systemintegration

Die Integration von JAXFront® in eine bestehende IT-Systemarchitektur unterscheidet sich massgeblich von der Wahl des eingesetzten Renderers.

Wie im Kapitel 2.2 beschrieben braucht es für das Rendering von JAXFront® Benutzeroberflächen zumindest ein XML Schema. Von wo dieses Schema beim Start von JAXFront® und über welchen Transportmechanismus (IIOP; HTTP, SOAP,...) diese geladen werden ist nicht relevant. Gleiches gilt für XUI Definitionen sowie eventuell vorhandenen XML Templates (XML Instanzen).

Eine Möglichkeit die XML Schemas, die XUI Definitionen und die XML Templates zu laden, ist die Verwendung eines Webservers, über welchen via HTTP URL die Ressourcen geladen werden. Eine andere Möglichkeit ist die Speicherung dieser Dokumente in einer XML- oder relationalen Datenbank.

Wird der JavaSwing Renderer von JAXFront® eingesetzt, so ist die Verfügbarkeit einer JavaVM auf jedem Rechner ein Muss. Die Integration geschieht dann meist in einem bereits vorhandenem Java Client Framework (Tier I). Beim HTML Renderer wird oft eine Integration in ein bereits bestehendes Server Framework (zB. Struts) auf dem mittleren Tier (Tier II) verlangt.

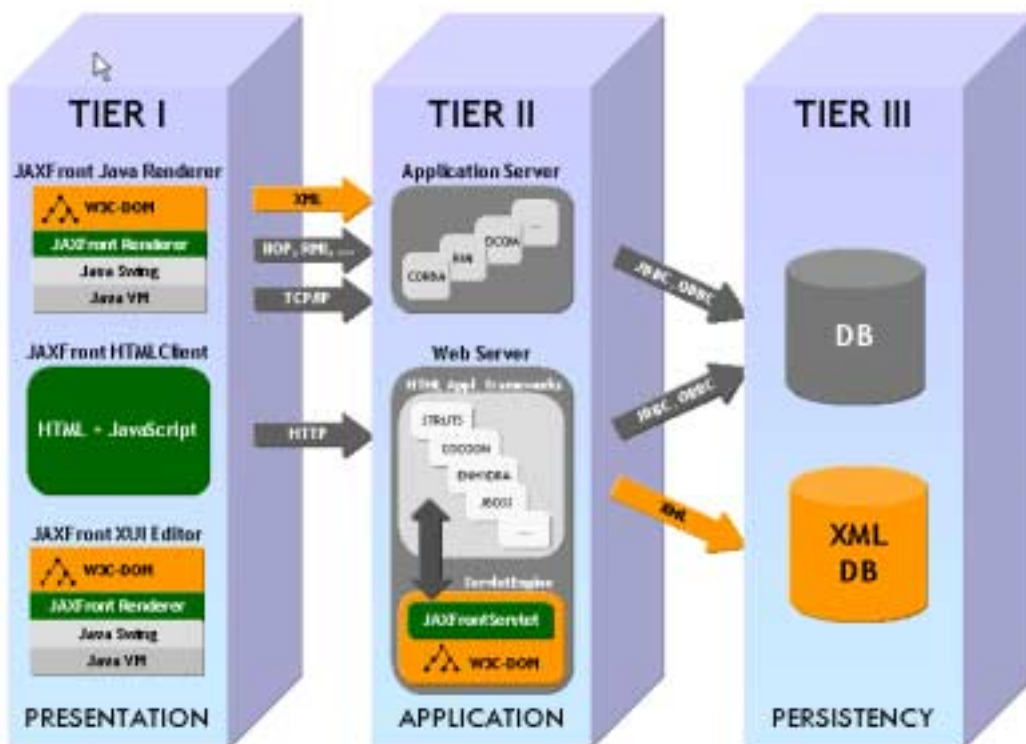



Abbildung 22: JAXFront® - 3tier Integration

 JAXFront® verwendet bewusst kein bestehendes client- oder serverseitiges Framework um möglichst offen resp. flexibel zu bleiben. Die Integration in ein bestehendes Framework ist jedoch sehr einfach und wird im folgenden Abschnitt genauer beschrieben.

5.1 Integration des JavaSwing Renderer

Der JavaSwing Renderer liefert für jeden Knoten aus einem XML Schema (der via XPath adressiert wird) die dazugehörige visuelle Java Komponente in der Form einer *javax.swing.JComponent* zurück.

Für den JAXFront® SwingRenderer ist nur eine Klasse von absoluter Wichtigkeit: *com.jaxfront.swing.ui.editor.EditorPanel*. Sie dient als Container (JPanel) für die darzustellenden XML Knoten. Erzeugt wird sie durch folgenden Konstruktor:

com.jaxfront.swing.ui.editor.EditorPanel

```
public EditorPanel(Type type, JFrame frame)
```

Erzeugt ein GUI Container (JPanel) für den darzustellenden Typen (XML Knoten).

Parameter	Beschreibung
Type	JAXFront® Type
Frame	Frame welches das EditorPanel hält.

Returnwert	Beschreibung
EditorPanel	GUI Container (JPanel) für den darzustellenden Typen (XML Knoten).



```
Document dom = DOMBuilder.getInstance().buildDocument(_url, "purchaseOrder");
Type rootType = dom.getRootType();

EditorPanel editorPanel = new EditorPanel(rootType, this);

getContentPane().setLayout(new BorderLayout());
getContentPane().add(editorPanel, BorderLayout.CENTER);
setSize(640, 480);
```

Abbildung 23: HelloWorld für JAXFront® (Codebeispiel)



Optional kann der Container für jeden JAXFront® Typen auch selbst erzeugt werden. Dazu dient die zentrale Factoryklasse um Java Swing Widgets zu erstellen: *com.jaxfront.core.ui.TypeVisualizerFactory*.

com.jaxfront.core.ui.TypeVisualizerFactory

```
public Visualizer getVisualizer(Type type)
```

Erzeugt für einen JAXFront® Typen eine visuelle Repräsentation in JavaSwing (JComponent).

Parameter	Beschreibung
Type	JAXFront® Type

Returnwert	Beschreibung
Visualizer	Eine konkrete Implementation des Visualizers Interface. Im Falle des JavaSwing Renderers eine JComponent.

Nachfolgendes Beispiel zeigt die Erzeugung eines eigenen GUI Containers (JPanel) und die Platzierung einzelner Komponenten auf dem Container:



com.jaxfront.helloworld.OwnContainer

```
Document dom = null;
String url = "c:\\jaxfront_swing\\examples\\po.xsd";
try {
    dom = DOMBuilder.getInstance().buildDocument(url, "purchaseOrder");
    Type rootType = dom.getRootType();

    getContentPane().setLayout(new BorderLayout());
    JPanel myContainer = new JPanel();
    myContainer.setLayout(new BoxLayout(myContainer, BoxLayout.Y_AXIS));
    JComponent shipToComponent =
        (JComponent)TypeVisualizerFactory.getInstance().getVisualizer(rootType.getChild("shipTo"));
    JComponent billToComponent =
        (JComponent)TypeVisualizerFactory.getInstance().getVisualizer(rootType.getChild("billTo"));
    myContainer.add(shipToComponent);
    myContainer.add(billToComponent);

    getContentPane().add(myContainer, java.awt.BorderLayout.CENTER);
    setSize(300,500);
}
catch (SchemaCreationException ex) {
    System.out.println("unable to create XML Schema from" + url);
}
catch (DocumentCreationException domEx) {
    System.out.println("unable to create JAXFront DOM from" + url);
}
```

Abbildung 24: Das Benutzen der TypeVisualizerFactory (Codebeispiel)

5.2 Integration des HTML Renderers

Die HTML Rendering Architektur von JAXFront® basiert auf der Java Servlet Technologie von SUN Microsystems. Herzstück bildet dabei ein JaxFrontServlet, welches für die Interaktion zwischen Server und Browser verantwortlich ist.

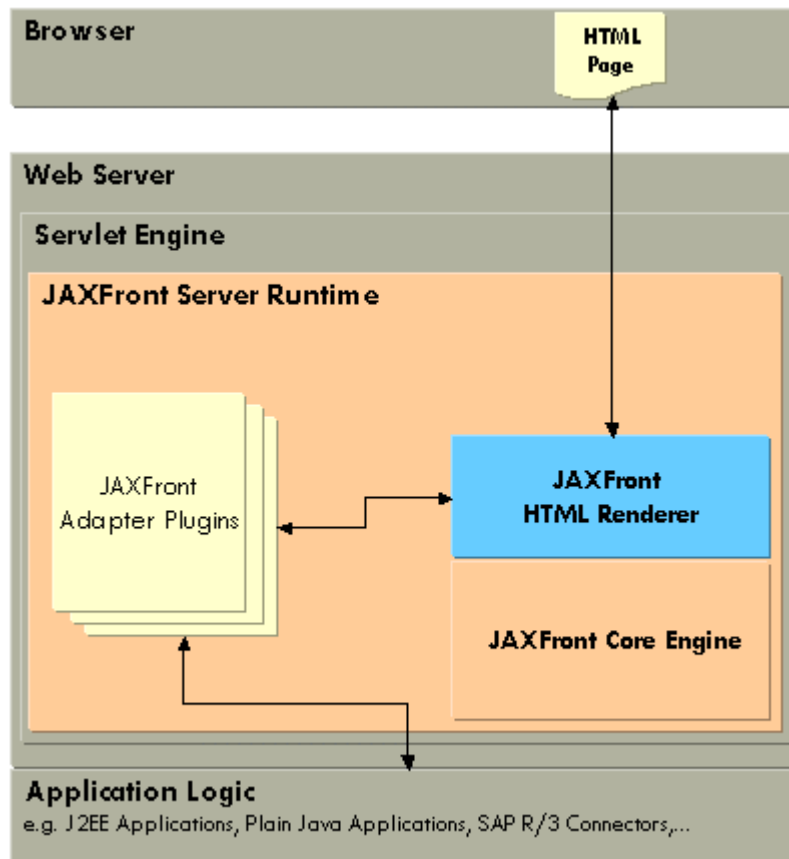


Abbildung 25: JAXFront® HTML Architektur

Mit Hilfe von JAXFront® Adapter Plugins (Java Klassen) können applikationsspezifische Erweiterungen definiert werden.

Der JAXFront® Server Adapter bietet einige Methoden um ein JAXFront® DOM zu erstellen, zu validieren oder zu speichern. Der Server Adapter kann durch eine eigene Implementation jederzeit erweitert werden.

Der Aufruf des JAXFront® Server Adapters erfolgt durch die Übergabe von Servlet Parametern.

Servlet-Übergabeparameter

Es sind folgende Server-Calls möglich, um das Servlet anzusteuern:

Parameter-Beschreibung

?jaxfront=load&xsd= "...Pfadangabe..."

Das Servlet erzeugt auf dem Server ein JAXFront® DOM gemäss XSD-Pfadangabe. Der Output ist eine HTML Oberfläche für das eben erstellte DOM.

Hinweis: Die Mitgabe eines Schema-Pfades ist zwingend, da ohne XML-Schema kein DOM gebildet werden kann.

Beispiel: ?jaxfront=load&xsd="c:/files/test.xsd"

?jaxfront=load&xsd= "...Pfadangabe..."&xml= "...Pfadangabe..."

Das Servlet erzeugt auf dem Server ein JAXFront® DOM gemäss Pfadangaben. Der Output ist eine HTML Oberfläche für das eben erstellte DOM.

Beispiel: ?jaxfront=load&xsd="c:/files/test.xsd"&xml="c:/files/test.xml"

?jaxfront=load&xsd= "...Pfadangabe..."&xml= "...Pfadangabe..."&xui= "...Pfadangabe..."

Das Servlet erzeugt auf dem Server ein JAXFront® DOM gemäss Pfadangaben. Der Output ist eine HTML Oberfläche für das eben erstellte DOM.

Beispiel:

?jaxfront=load&xsd="c:/files/test.xsd"&xml="c:/files/test.xml"&xui="c:/files/test.xui"

5.3 Integration des PDF Renderers

Jedes JAXFront® DOM kann direkt im PDF-Format gerendert werden, falls die JAXFront® PDF-Library installiert wurde. Dabei spielt die Klasse PDFGenerator die zentrale Rolle um aus einem JAXFront® DOM eine PDF-Datei zu erstellen.

com.jaxfront.pdf.PDFGenerator

```
public ByteArrayOutputStream print(Document)
public ByteArrayOutputStream print(Document, Document)
```

Erzeugt aus dem aktuellen DOM einen binären PDF Outputstream. Falls kein PDF-XUI übergeben wird (2. Paramter) wird das dazugehörige PDF-XUI über die Fingerprint Information des aktuellen DOM's gesucht. Falls keines vorhanden ist, kommt das Default Rendering zum Zuge.



Beispiel, um aus einem DOM eine PDF-Datei zu erzeugen:

```
Document dom = DOMBuilder.getInstance().buildDocument(_url, "purchaseOrder");
ByteArrayOutputStream bos = PDFGenerator.getInstance().print(dom);

if (bos != null) {
    try {
        String tempPDFName = "c:\\temp\\test.pdf";
        FileOutputStream fos = new FileOutputStream(tempPDFName);
        bos.writeTo(fos);
        fos.close();
    }
    catch (Throwable t) {}
}
```

6 *Wichtige Links*

Dieses Dokument	http://www.jaxfront.com/download/jaxfront-konzepte-1_50.pdf
W3C	http://www.w3.org
XML	http://www.w3.org/TR/REC-xml
XML Schema 1.0	http://www.w3.org/TR/xmlschema-0/
XML Schema Best Practice Homepage	http://www.xfront.com/BestPracticesHomepage.html
XPath 1.0	http://www.w3.org/TR/xpath
JAXFront® Homepage	http://www.jaxfront.com
JAXFront® Online Demo	http://www.demo.jaxfront.com
JAXFront® Q&A Forum	http://www.jaxfront.com/cgi-bin/forum/YaBB.pl
xcentric technology & consulting	http://www.xcentric.ch