technische universität dortmund

IT in Produktion und Logistik

Techniken zur Generierung dynamischer Portalinhalte in Logistikunternehmen

Studienarbeit

aus dem Gebiet IT in Produktion und Logistik

Autor: Florian Lechner, Matrikelnummer: 112479, florian-lechner@gmx.de

Studienrichtung: Dipl. Angewandte Informatik (Anwendungsfach Logistik)

Lehrstuhl: 7 – Maschinenbau

Betreuer: Univ.-Prof. Dr.-Ing. Markus Rabe

Zweiter Betreuer: Dipl.-Inf. Anne Antonia Scheidler

Abgabedatum: 15.Juni 2012

Inhalt:

I. Einleitung

a) Vorwort und Aufgabenstellung

Im Zuge der fortschreitenden Unternehmensvernetzungen gewinnen Webportale für den Bereich Logistik zunehmend an Bedeutung. Insbesondere die Koordination verschiedener Zulieferbetriebe sowie die damit verbundene Auftragsabwicklung finden immer häufiger unter Zuhilfenahme informationstechnischer Möglichkeiten statt.
Ziel der Arbeit ist die Erstellung eines Konzeptes zur Umsetzung dynamischer Portalinhalte. Hierbei werden zuerst die zu erfassenden Entitäten in einer Beschreibungssprache strukturiert. Im Anschluss wird ein Verfahren aufgezeigt, das aus der Entitätenbeschreibung eine webbasierte Benutzermaske erzeugt. Die über diese Eingabemaske übermittelten Daten werden unter Zuhilfenahme einer Scriptsprache in einer Datenbank gespeichert. Für diese Schritte werden Regelkonzepte vorgestellt, die das problemlose Zusammenspiel zwischen den Technologien und den einzelnen Programmteilen sicherstellen sollen. Im letzten Kapitel wird eine kleine exemplarische Umsetzung des Konzepts mit einigen ausgewählten Regeln vorgestellt.

Das in dieser Arbeit entworfene Konzept ist unabhängig von den später verwendeten Techniken XML, PHP, DTD und MySQL. Insbesondere für XML-Dateien existieren eine Vielzahl von frei verfügbaren Parsern, allerdings enthält PHP in der aktuellen Version 5 genügend Funktionen um die in dieser Arbeit auftretenden Anforderungen auf unkomplizierte Weise zu erfüllen.

Der "State of the Art" ist in dieser Arbeit implizit durch die verwendeten Techniken bzw. Versionen von XML, PHP und MySQL enthalten, die in Kapitel III. und V. verwendet werden.

b) Definitionen

Begriff	Abkürzung	Beschreibung
Entitätsstruktur /XML-Datei	ES	Die Datei, die die strukturierte Beschrei- bung der Entitäten enthält, in dieser Arbeit in der Sprache XML.
Eingabemaske	EM	Die webbasierte Maske für die Eingabefelder, die der Nutzer im Browser angezeigt bekommt.
Strukturdefinition/DTD-Datei	SD, DTD	Die Strukturdefinition bezeichnet die Vorschriften zur Strukturierung der Entitätsstruktur. In dieser Arbeit per Document-Type-Definition in einer separaten Datei.
Eingabedaten		Die Eingaben des Nutzers in die Felder der Eingabemaske.

II. Allgemeines Konzept und Funktionsweise

a) Beschreibung des Gesamtkonzepts

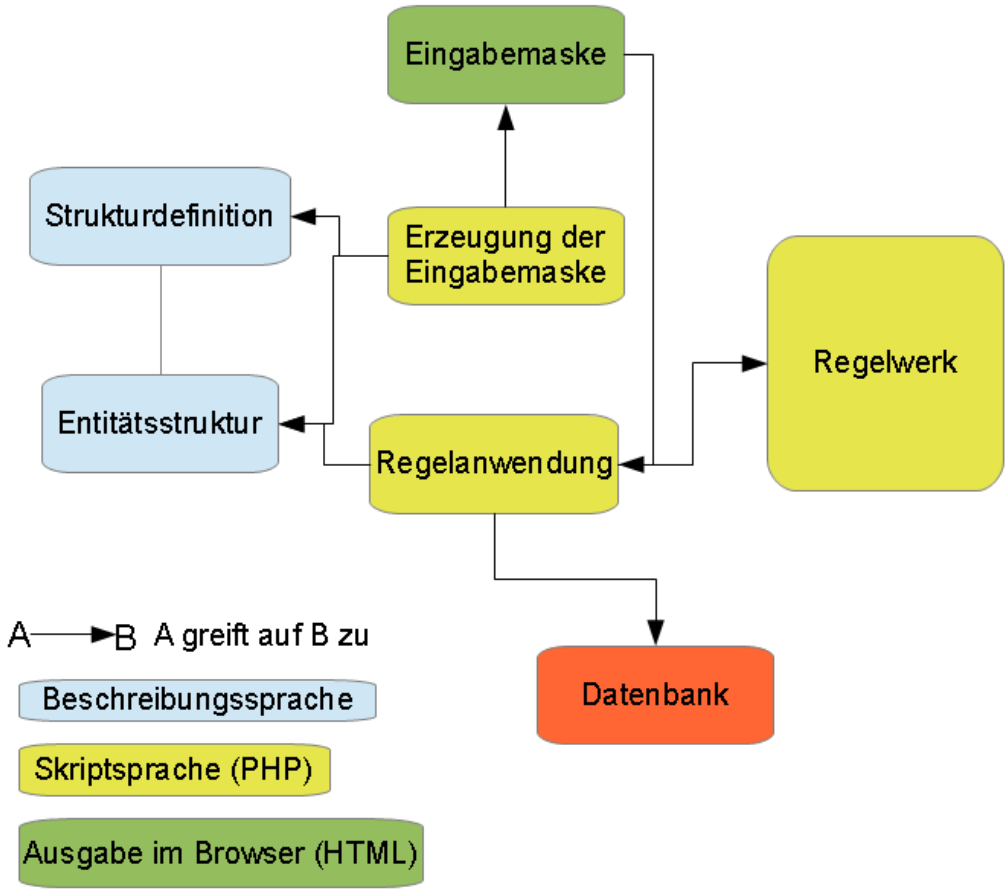

Abb 1: Skizze des grundlegenden Konzepts

Das Konzept besteht allgemein aus folgenden Objekten, die hier in der konzeptionellen Sicht zum besseren Verständnis ihrer Funktionsweise und ihres Zusammenspiels als einzelne Objekte dargestellt werden. In der späteren technischen Umsetzung müssen diese Objekte aber nicht zwingend in einzelne Dateien bzw. Programme unterteilt

werden.

- Entitätsstruktur (ES): Dies ist die Grundlage des Konzepts. Die ES ist eine strukturierte Beschreibung der Eingabemaske. Sämtliche Elemente die für die Erzeugung der Eingabemaske nötig sind, werden hier gespeichert. D.h. alle Entitäten die erfasst werden sollen, ihr Aufbau sowie deren Eigenschaften wie beispielsweise der Name. Außerdem die Beziehung der einzelnen Entitäten untereinander (Abhängigkeiten, Untergruppen etc.). Diese Struktur dient als Vorlage für die Eingabemaske und kann auch zur Erstellung der Datenbank als Vorlage dienen (wie in Kapitel III. c) erläutert) . Die ES ist der Ausgangspunkt des Hauptprogramms und der Teil, der später von den Nutzern des Programms am häufigsten eingesehen und verändert wird. Der genaue Aufbau wird in Kapitel III a) ausführlicher beschrieben.

- Strukturdefinition: Hier werden die Wertebereiche, die erlaubten Zeichen sowie die Formate der einzelnen Entität-Typen definiert. Diese Definitionen werden später von den Regelwerken benötigt, um einige Regeln anzuwenden. Beispielsweise wird hier das Format einer e-mail-Adresse beschrieben, oder die erlaubten Zeichen einer Kundennummer festgelegt. Außerdem kann es später bei der Erzeugung bzw. Darstellung der Eingabemaske sinnvoll sein, auf die Strukturdefinitionen zurückzugreifen um z.B. statt einem Eingabefeld für bestimmte Wertebereiche direkt ein Drop-Down-Menue anzubieten, welches sich die angebotenen Werte direkt aus der Strukturdefinition holt. Die Strukturdefinition bildet einen Zusatz zur Entitätsstruktur. Außerdem kann in der Strukturdefinition auch festgelegt werden, aus welchen Entitäten die Entitätsstruktur bestehen darf, wie die Entitäten aufgebaut sind (welche Attribute sie haben) und wie viele Entitäten jeweils vorkommen dürfen bzw. müssen. So kann zumindest teilweise verhindert werden, dass die Entitätsstruktur unsinnigen Inhalt enthält. Je nach gewählter Sprache für die spätere Implementierung können die Strukturdefinitionen auch in der Entitätsstruktur direkt enthalten sein. In dieser Arbeit werden für die Strukturdefinition und die Entitätsstruktur zwei getrennte Objekte verwendet um sich so stark wie möglich an der vorgestellten Konzeptionellen Sicht zu orientieren.

- Skript zur Erzeugung der Eingabemaske: Dieser Programmteil erzeugt die Eingabemaske die dem Nutzer angezeigt wird. Hierzu wird auf die Struktur-Definitionen und die Entitätsstruktur zugegriffen.

- Eingabemaske: Die Eingabemaske, die anhand der Strukturdefinition erzeugt wurde und dem Nutzer angezeigt wird. Nach der Eingabe der Daten und dem "Absenden" wird das Skript zur Regelanwendung aufgerufen und diesem die Eingabedaten übermittelt.

- Skript zur Regelanwendung: Neben dem Programm zur Erzeugung der Eingabemaske ist dies der Hauptbestandteil des Konzepts und mit der wichtigste Schritt im gesamten Ablauf. Hier werden für jedes Objekt der Eingabemaske die jeweiligen Regeln angewendet und die Eingaben des Nutzers überprüft. Der genaue Aufbau und Ablauf dieser Operationen wird in Kapitel III b) beschrieben.

- Regelwerk: Das Regelwerk enthält alle Regeln die auf die verschiedenen Eingaben

des Nutzers angewendet werden können. Hier gibt es verschiedene Möglichkeiten wie die Regeln organisiert werden und wie genau sie aufgebaut sind. Auf diese Variationsmöglichkeiten und ihre Umsetzung wird in Kapitel IV genauer eingegangen.

– Datenbank: Nachdem sie durch die Regelanwendung überprüft und ggf. gefiltert wurden, werden die Daten aus der Eingabemaske hier gespeichert. Außerdem enthält die Datenbank evtl. schon vorhandene Datensätze, die unter Umständen durch einige Regeln abgerufen werden können. Weitere Informationen zur Datenbank werden in Kapitel III d) und IV d) dargestellt. Allerdings liegt der Schwerpunkt dieser Arbeit nicht auf der Datenbank und ihrer Funktionsweise.

Die obige Beschreibung stellt die Grundelemente des Konzepts dar. Allerdings gibt es an verschiedenen Stellen diverse Möglichkeiten in der Ausgestaltung des Ablaufs und der genauen Funktion der einzelnen Schritte. Diese werden nun erläutert und hinsichtlich ihrer Vor- und Nachteile verglichen.

Als erstes stellt sich die Frage, wo und wie festgelegt und gespeichert wird, welche Regeln für welche Daten angewendet werden. Hierbei gibt es folgende Möglichkeiten:

– Manuelle Angabe der Regeln: Alle Regeln die für ein spezielles Eingabefeld bzw. Interaktionsobjekt der Eingabemaske angewendet werden sollen, müssen explizit in der Entitätsstruktur (z.B. als Attribut) angegeben werden. D.h. es muss der Name der anzuwendenden Regel inklusive aller nötigen Parameter vorhanden sein.
Dadurch ergeben sich folgende Vorteile:
- Diese Vorgehensweise gibt dem Nutzer die größtmögliche Kontrolle über die Anwendung der Regeln. Alle Regeln, die für ein Eingabefeld in Frage kommen, können angewendet werden. So werden so viele Fehler wie möglich erkannt und abgefangen.
- Außerdem gibt es bei dieser Methode die größte Flexibilität was die Gestaltung der Entitäten betrifft: Enthält das Regelwerk genügend Regeln, die nicht zu "grobkörnig" filtern, so kann man durch eine geeignete Auswahl exakt die erlaubten Eingabezeichen bzw. Formate beschreiben, die für die jeweilige Entität vorgesehen sind.

Allerding ergeben sich bei dieser Methode einige nicht unerhebliche Nachteile:

- Zum Einen wird vom Nutzer ein hohes Maß an Kenntnis über die Regeln und deren Funktionsweise verlangt. Jedes mal, wenn der Nutzer eine neue Entität einfügen will, oder eine bestehende verändert, müssen die passenden Regeln angegeben werden. Dies bedeutet, dass jede Regel bekannt sein muss. Werden zu wenige Regeln angegeben, bleiben evtl. fehlerhafte Eingaben unbemerkt.

- Zum anderen muss für jede angegebene Regel geprüft werden, ob diese auch für die jeweilige Entität anwendbar ist. Dies muss entweder vom Nutzer selbst sichergestellt werden, oder im Programmteil, welches die Überprüfung der Eingabedaten mit Hilfe dieser Regeln vornimmt.

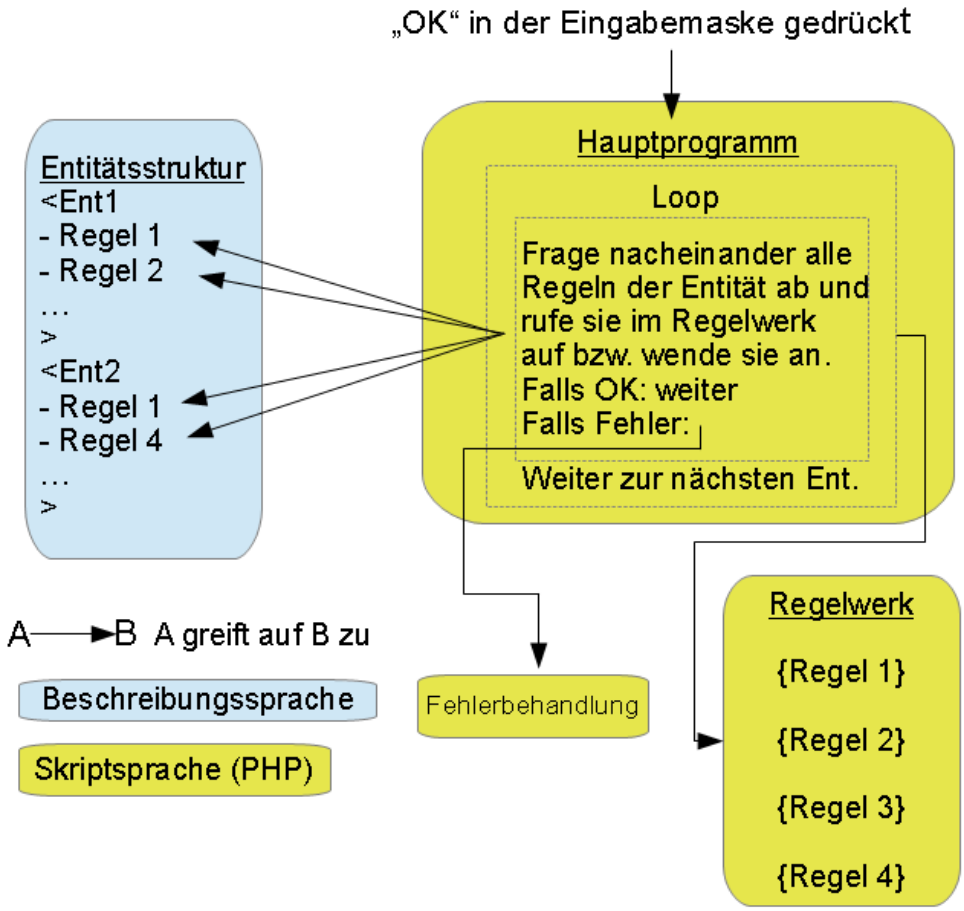

Abb 2: Skizze zur manuellen Angabe der Regeln

– Automatische Anwendung der Regeln: Die Auswahl welche Regeln angewendet werden erfolgt automatisch, ohne dass der Nutzer einzelne Regeln angeben muss bzw. kann. Dies kann z.B. realisiert werden, indem für jede Entität ein Typ angegeben wird. Für ein Textfeld, in das der Nutzer ein Kommentar schreiben kann, könnte dies beispielsweise der Typ "String" sein. Nun kann im Programmteil, das die Regeln anwendet eine Automatisierung erfolgen, die bewirkt, dass alle Regeln die z.B. unerlaubte Sonderzeichen herausfiltern auf jede Entität vom Typ "String" angewendet werden. Durch diese Automatisierung ergeben sich folgende Vorteile: - Es kann erreicht werden, dass zumindest die einfachen und grundlegenden Regelüberprüfungen automatisiert werden. Dadurch wird die Entitätsstruktur übersichtlicher und einfacher zu handhaben, da hier nicht mehr die einzelnen Regeln mit angegeben werden müssen.

- Außerdem muss der Nutzer nun nicht mehr die Regeln kennen und sich auch nicht mehr damit befassen, wo und wie diese angegeben werden müssen. Lediglich der gewünschte Typ muss einmal in der Definition der Entität angegeben werden.

Allerdings entstehen durch diese Methode auch einige Nachteile: - Es ist relativ kompliziert das System um neue Typen zu erweitern, da nicht nur die Entitätsstruktur bearbeitet werden muss, sondern auch der Programmcode, der für die Anwendung der Regeln zuständig ist bzw. das Regelwerk, welches den verschiedenen Typen die jeweiligen Regeln zuordnet. -
Ein weiterer Nachteil ist die fehlende Möglichkeit bestehende Typen geringfügig anzupassen ohne ein komplett neuen Typ zu definieren: Angenommen der Nutzer benötigt zwei String-Typen, wobei der erste alle Buchstaben enthalten darf, der zweite zusätzlich noch das "@"-Zeichen, dann werden hierfür zwei Typen definiert, obwohl sie sich nur um ein einziges Zeichen unterscheiden. Dies kann unter Umständen zu einer riesigen und unübersichtlichen Ansammlung von Typen führen.

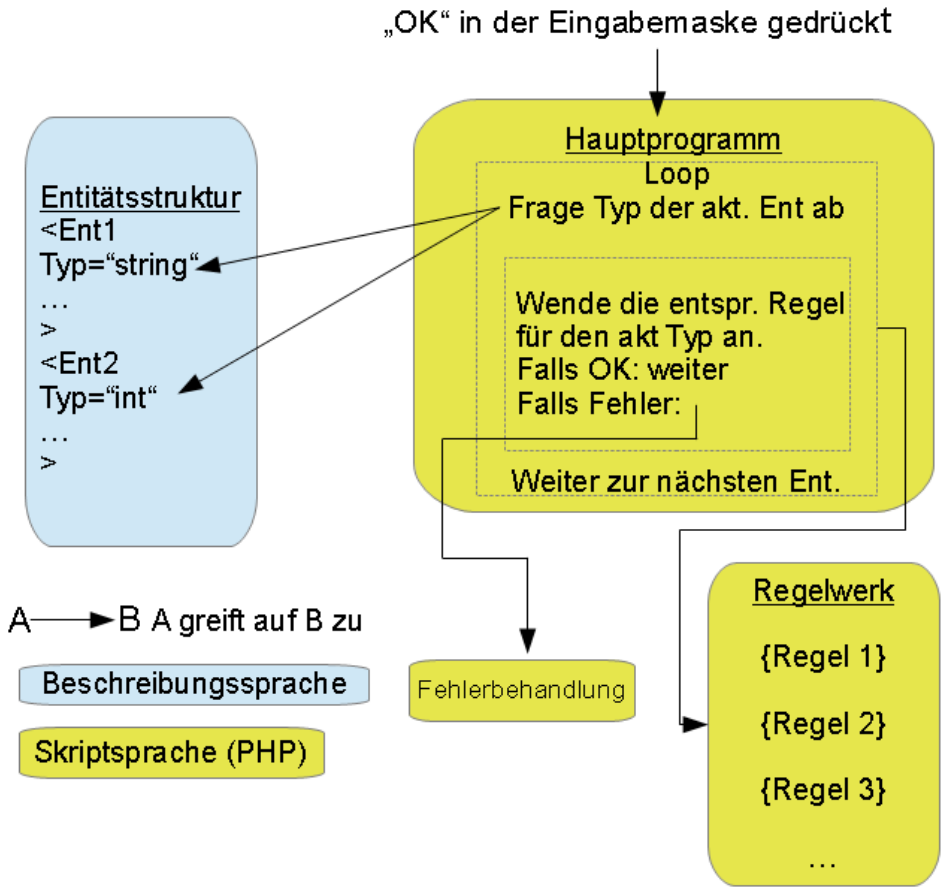

Abb 3: Skizze zur automatischen Angabe der Regeln

- Automatische und manuelle Anwendung der Regeln: Die wohl komfortabelste und effizienteste Methode ist eine Mischung aus den beiden eben vorgestellten Methoden. Das Ziel bei dieser Variante ist es, die Vorteile beider bisher genannten Varianten zu vereinen. Dabei wird die automatische Anwendung der Regeln um die zusätzliche optionale Möglichkeit erweitert, weitere Regeln anzugeben. Wird als Typ noch ein sog. "Null-Typ" definiert, der keine Regeln beinhaltet, kann durch dessen Angabe im Typ-Attribut der Entität auch die Variante der manuellen Angabe der Regeln simuliert werden. Die Vorteile einer solche Lösung wären folgende (zusätzlich zu den oben genannten): - Einfachen und häufig verwendeten Typen können vordefiniert werden. So kann ein Großteil der Arbeit vereinfacht und automatisiert werden, ohne sich gleichzeitig auf diese Typen beschränken zu müssen. - Der Nutzer kann selbst entscheiden, ob und wie weit er die Anwendung der Regeln mitbestimmt. Von der vollständigen Automatisierung mit Hilfe eines vordefinierten Typs, bis hin zu manuellen Angabe sämtlicher Regeln, sind alle Abstufungen möglich. - Innerhalb der Entitätsstruktur kann problemlos zwischen den Varianten gewechselt werden. Während die erste Entität nur mit Hilfe eines vordefinierten Typs überprüft wird, kann die nächste bei Bedarf über die manuelle Angabe der Regeln geprüft werden.

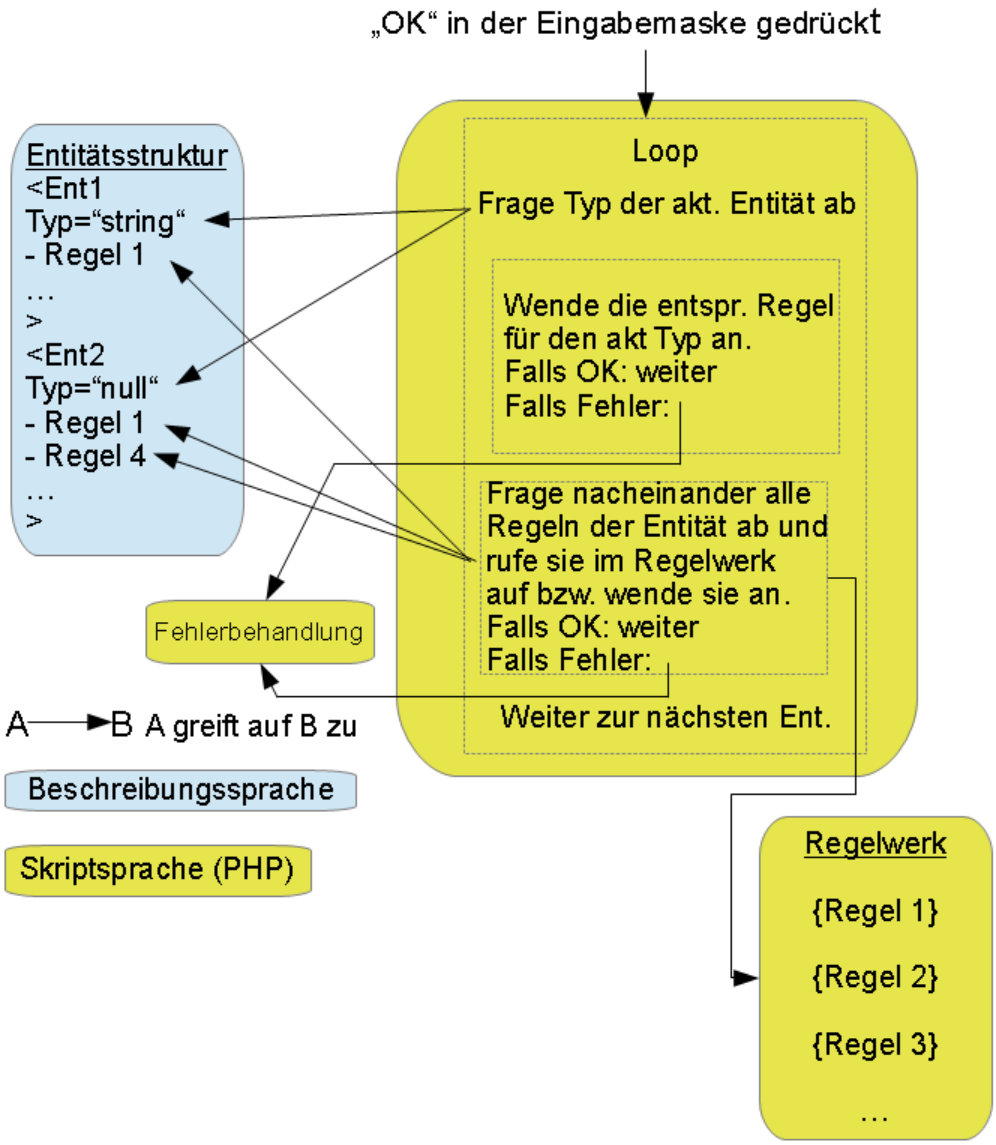

Abb 4: Skizze zur automatischen und manuellen Angabe der Regeln

Was die Konzeption des Regelwerkes betrifft, so gibt es auch hier einige mögliche Varianten wie die Regeln organisiert, gruppiert und beschrieben werden können. Dies wird in Kapitel IV a) genauer besprochen.

III. Beschreibungssprache und Transfer zur Eingabemaske

a) Beschreibungssprache

In diesem Kapitel wird der technische Aufbau der ES und der SD erläutert. Wie anfangs erwähnt wird in dieser Arbeit XML (Version 1.0) für die ES und PHP (Version 5.3.8) als Skriptsprache verwendet. Für die SD wird Document-Type-Definition (DTD) verwendet.

Um den Aufbau und die Funktionsweise der ES und SD anschaulicher zu erläutern, wird im Folgenden angenommen, dass der Nutzer eine Kundennummer eingeben soll und die gewünschte Anzahl der zu bestellenden Exemplare eines Produkts mit dem Namen "Testprodukt". In der ES muss nun als erstes festgelegt werden, dass es sich um eine XML-Datei handelt und deren Art der Codierung, dies sieht dann z.b. so aus:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Als nächstes muss der Name und Speicherort der verwendeten DTD angegeben werden. Es ist in XML zwar auch möglich, die DTD in die XML-Datei selbst zu schreiben, aber die Verwendung einer externen DTD ist nicht nur übersichtlicher, sondern orientiert sich auch stärker am hier vorgestellten Konzept. Mit folgender Zeile wird festgelegt, dass die DTD in der Datei "entdef.dtd" zu finden ist, und sich im selben Verzeichnis wie die XML-Datei befindet.

```
<!DOCTYPE entstruc SYSTEM "entdef.dtd">
```

Da hier eine wohlgeformte XML-Datei verwendet werden soll, müssen alle Entitäten (in diesem Fall die Kundennummer und das Testprodukt) in einer äußeren Entität gekapselt werden. Der Name dieser äußeren Entität spielt keine Rolle und lautet in diesem Fall "entstruc". Innerhalb dieser äußeren Entität werden nun die Strukturen für die eigentlichen Entitäten beschrieben. In diesem Fall also die Kundennummer und das Testprodukt:

```
<entstruc>

  <kundennummer id="ID1"/>

  <produkt id="ID2" name="testprodukt"/>

</entstruc>
```

Damit wären zwei Entitäten beschrieben, die als "kundennummer" und "produkt" bezeichnet sind. Für die weitere Verarbeitung im nachfolgenden Programm ist es wichtig, dass alle Entitäten außerdem ein Attribut besitzen, in dem sich ihre ID speichern lässt, also der einzigartige Wert zur Identifizierung. Der Wert der ID-Attribute aller Entitäten wird später im Programmablauf anhand einer Regel automatisch gesetzt. Dabei werden Werte die von früheren Programmaufrufen oder von manuellen Einträgen durch den Nutzer stammen überschrieben. Das Produkt erhält außerdem noch ein weiteres Attribut um den Namen des Produktes zu speichern der später in der EM zu sehen ist. Es lassen sich auf diese Weise viele weitere Entitäten für die Eingabemaske definieren, z.b. ein Kommentarfeld für Anmerkungen:

```
<kommentar id = "ID3" />
```

oder ein komplexeres Produkt:

```
   <produkt id="ID4" name="testprodukt" farbe = "schwarz" groesse
= "XL" ausfuehrung = "kompakt"/>
```

Eine sinnvolle und funktionierende ES für die angestrebten Beispieldaten sieht dann so aus:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE entstruc SYSTEM "entdef.dtd">

<entstruc>

  <kundennummer id="ID1"/>

  <produkt id="ID2" name="testprodukt"/>

</entstruc>
```

Ohne die DTD ist allerdings auch eine Entität mit falschen oder fehlenden Attributen oder unerlaubter Verschachtelung möglich:

```
<produkt  unnoetigesAttribut = "Blubb"/>
```

Hier fehlt sowohl der Name als auch die zwingend erforderliche ID, außerdem ist ein überflüssiges Attribut vorhanden.

Um sicherzustellen, dass solche Entitäten nicht in die ES geschrieben werden können, wird in der DTD genau beschrieben, welche Entitäten erlaubt sind, welche Attribute sie besitzen und wie oft sie vorkommen dürfen.

Mittels den folgenden Zeilen wird in der DTD festgelegt, dass das Element "entstruc" das äußere Wurzelelement ist, und die Elemente "kundennummer" und "produkt" enthält und zwar in dieser Reihenfolge. Dabei muss "kundennummer" genau einmal vorkommen und "produkt" kann beliebig oft vorkommen.

```
<!ELEMENT entstruc (kundennummer, produkt)>

<!ELEMENT kundennummer (#PCDATA)>

<!ELEMENT produkt (#PCDATA)*>
```

In einer DTD lassen sich noch viele weitere und komplexere Strukturierungen vornehmen, für dieses Beispiel reichen die oben verwendeten Definitionen aus. Natürlich lassen sich für die SD auch alternative Techniken wir z.b. XSD verwenden oder eine komplett selbst definierte SD. Wichtig ist nur, dass überhaupt eine SD

vorhanden ist, denn mit Hilfe einer solchen SD lassen sich auf relativ einfache Weise eine Menge an unnötigen und ungewollten Entitäten in der ES verhindern. Außerdem ist eine Implementierung des Konzepts später um einiges einfacher, wenn die möglichen Entitäten vollständig bekannt und definiert sind.

Als nächstes muss noch festgelegt werden, welche Attribute die jeweiligen Entitäten haben und von welchem Typ die Attribute sind. Mit den folgenden Zeilen wird festgelegt, dass das Element "produkt" einen Namen und eine ID besitzen muss und die Kundennummer nur eine ID. Das Schlüsselwort #REQUIRED bedeutet, dass dieses Attribut enthalten sein muss.

```
<!ATTLIST produkt

name      CDATA       #REQUIRED

id        ID       #REQUIRED>

<!ATTLIST kundennummer

id        ID        #REQUIRED>
```

Mit Hilfe solcher DTDs kann relativ gut gesteuert werden, wie die ES auszusehen hat. Eine Vielzahl unsinniger und unvollständiger Entitätsbeschreibungen werden so abgefangen. Natürlich kann es auch vorkommen, dass die SD vom Nutzer geändert wird, und so Entitäten in der ES erlaubt werden, die eigentlich so nicht vorgesehen waren. Um dem vorzubeugen kann die SD (die in diesem Fall als einzelne Datei vorliegt) mit entsprechenden Zugriffsrechten versehen werden und/oder in ein geschütztes Verzeichnis oder sogar auf einen anderen Server ausgelagert werden. Somit hätten nur ausgewählte Personen Zugriff auf die DTD, während der "normale" Nutzer weiterhin die ES ändern kann, sich aber an die DTD-Vorgaben halten muss.

b) Erstellung der Eingabemaske

Bevor die Eingabemaske erstellt werden kann, müssen alle Regeln, die die ES auf Korrektheit überprüfen, angewendet werden. Diese Regelgruppe wird in Kapitel IV b) beschrieben.

Als erstes muss also geprüft werden, ob die ES (die in diesem Fall als XML-Datei vorliegt) vorhanden ist und geladen werden kann. Beim Laden der Datei wird somit auch gleich geprüft, ob die Syntax korrekt ist. Danach folgt die Validierung durch die DTD. Die ersten beiden Schritte sind in jeder Sprache die für das Programm in Frage kommt problemlos möglich. Für die Validierung existiert in PHP die "PHP DOM API“ mit der es sehr einfach ist, diese Validierung auszuführen. Es existieren aber auch viele frei verfügbare Programme um eine XML-Datei mit einer entsprechenden DTD zu prüfen.

Nachdem nun sichergestellt ist, dass mit der ES gearbeitet werden kann, und diese eingelesen bzw. geladen wurde, muss nun als erstes geprüft werden, ob Entitäten existieren, die gleiche Namen haben. Rein theoretisch lässt sich auch mit gleichnamigen Entitäten weiterarbeiten, da diese über ihre einzigartigen ID-Attribute angesprochen

werden, aber gleichnamige Entitäten würden spätestens bei der Eingabemaske für Verwirrung sorgen. Eine einfache Methode zur Überprüfung auf gleiche Namen sieht wie folgt aus:

Die ES wird durchlaufen und für die erste Entität mit einem Namens-Attribut wird dieses zwischengespeichert. Danach werden in einer Schleife alle anderen Entitäten durchlaufen und auf evtl. vorhandene Namens-Attribute mit gleichem Wert wie der zuvor gespeicherte geprüft. Sobald sich ein Treffer ergibt wird abgebrochen und eine entsprechende Meldung für doppelte Namen ausgegeben. Ansonsten wird der Name der nächsten Entität zwischengespeichert und wieder die Schleife durchlaufen. Ergibt sich für keine der Entitäten ein Treffer, so ist sichergestellt, dass alle Namen nur einmal vorkommen und es kann mit dem nächsten Schritt im Programmablauf begonnen werden.

Nun müssen die ID-Attribute so gesetzt werden, dass sie einzigartig sind. Da jede Entität (wie in der DTD festgelegt) ein solches ID-Attribut besitzt, legt man am besten eine Variable mit dem Startwert 1 an und durchläuft alle Entitäten der ES. Dabei wird bei jedem Schleifendurchlauf der ID-Wert der aktuellen Entität auf den der Variable gesetzt und die Variable um 1 erhöht. Danach sind alle IDs aufsteigend und somit einzigartig.

Da natürlich auch weitere Entitäten der ES hinzugefügt werden können, bzw. vorhandene Entitäten gelöscht werden können, kann sich die ES zwischen zwei Programmabläufen geändert haben. Deshalb ist es wichtig, dass die Regel für das Setzen der IDs immer aufgerufen wird, bevor die Eingabemaske erstellt wird. Dadurch wird sichergestellt, dass auch nach Änderungen an der ES die ID-Werte einzigartig sind.

Als nächstes folgt die eigentliche Erzeugung der Eingabemaske. Auch hier wird die ES per Schleife durchlaufen. Für jede Entität wird zuerst der Typ abgefragt, der entweder in einem Attribut gespeichert ist, oder sich wie in unserem Beispiel direkt aus dem Namen (nicht dem Namens-Attribut!) der Entität (bzw. des XML-Knotens) ergibt. Für die Entität die in dem obigen Beispiel das Produkt mit dem Namen „Testprodukt“ darstellt, die der XML-Knoten so aus:

```
<produkt id="ID2" name="testprodukt"/>
```

Es wird also nach dem Knotennamen "produkt" gefragt, nicht nach dem Attributsnamen "Testprodukt". Nun kann je nach Name eine entsprechendes Objekt für die Eingabemaske erzeugt werden. Bei "produkt" ein Textfeld um die gewünschte Anzahl einzugeben, bei "expresslieferung" z.B. eine Checkbox oder bei "lieferdatum" ein Drop-Dow-Menue mit einer Datumsauswahl. Dazu lässt sich auch der Wert des Namens-Attributes auslesen und vor das Textfeld schreiben. Somit ergibt sich für die Entität

```
<produkt id="ID2" name="testprodukt"/>
```

folgender HTML-Code:

```
<INPUT TYPE=text name=$tmp value =""">
```

Die Variable "$tmp" hat den Wert des ID-Attributes, damit die Input-Objekte später eindeutig zugeordnet werden können.

Sind alle Entitäten durchlaufen und abgearbeitet, wird noch ein "Absenden"-Button hinzugefügt und die Eingabemaske ist vollständig.

c) Transfer und Prüfung der Daten

Für diesen Abschnitt wird angenommen, dass die Eingabemaske aus zwei Textfeldern besteht, das erste für die Kundennummer mit der ID "ID1" und dem eingegebenen Wert "DE1" und das zweite für das Testprodukt mit der ID "ID2" und dem eingegebenen Wert "12". Außerdem wird das Konzept für Regelklassen "automatische Anwendung der Regeln" aus Kapitel II a) verwendet.

Nachdem der Nutzer die Eingabefelder ausgefüllt hat und den "Absenden"-Button gedrückt hat, werden die Eingabedaten zuerst auf unerlaubte Zeichen u.ä. Geprüft. Diese Regeln werden in Kapitel IV a) als Gruppe 1 beschrieben.

In HTML werden die Eingabedaten in einem Feld gespeichert und durch ihr "name"-Attribut in der Iinput-Definition (<INPUT TYPE=text name=$tmp value ="">) angesprochen. Dieses Attribut wurde durch das Programm auf die entsprechende ID des Entität in der ES gesetzt.

Das Feld mit den Eingabedaten wird nun durchlaufen. Im Folgenden wird das Feld mit POST [ ], der Index mit i und dessen Wert (also die Eingabe im entsprechenden Textfeld) mit x bezeichnet. Für jedes i muss nun die ES durchsucht werden um die Entität mit dieser ID zu finden.

POST[ID1] hat den Wert "DE1", die Entität mit dem ID-Wert "ID1" ist:

```
<kundennummer id="ID1"/>
```

Von der Entität wird nun der Name (Knoten-Name des XML-Knotens) abgefragt, also "kundenummer" und die Regel für Kundenummern gestartet. Diese Regel erhält als Parameter nun die Eingabedaten im Textfeld der Kundennummer, also x mit dem Wert "DE1". Mit der Regel für die Kundenummer kann nun geprüft werden, ob die Eingabe einem gültigen Format für Kundennummern entspricht. Beispielsweise ob die Eingabe mit "DE", "EU" oder "US" anfängt um deutsche, europäische oder amerikanische Kunden zu unterscheiden.

Entspricht die Eingabe einer Kundennummer, dann liefert die Regel "true" zurück, ansonsten "false" und kann ein entsprechender Hinweis ausgegeben werden. Weitere Rückgabetypen werden in Kapitel IV a) besprochen.

Bei Produkten müsste geprüft werden, ob die Eingabe eine positive ganze Zahl ist, bei mail-Adressen, ob die Eingabe einem e-mail-Format entspricht usw.

Falls die Eingabedaten diese erste Überprüfung bestehen, wird eine Verbindung mit der Datenbank aufgebaut um die Eingabedaten mit evtl. vorhandenen Einträgen in der Datenbank abzugleichen. Dies ist u.a nötig, um zu verhindern, dass vorhandene Datensätze überschrieben werden, oder Tabellen oder Spalten in der Datenbank fehlen. Außerdem kann geprüft werden, ob die Kundennummern überhaupt existieren oder ob sich der Kunde erst registrieren muss.

Für das hier verwendete Beispiel würde die Tabelle "Bestellung" mit den Spalten "Kundennummer" und "Testprodukt" genügen, wobei "Kundennummer" als Primärschlüssel dient. Alternativ lassen sich die Spalten auch nach den jeweiligen IDs der Entitäten benennen, so müsste bei der Suche nach der passenden Spalte nicht unterschieden werden, ob die Entität ein namens-Attribut hat und die Spalte damit gekennzeichnet ist, oder ob der Name der Spalte dem Namen des XML-Knotens entspricht (da z.B. "kundennummer" keinen eigenes Namens-Attribut besitzt).

Der nächste Schritt ist vom Ablauf her identisch mit der Überprüfung auf unzulässige Zeichen und Formate der Eingabedaten: Für jeden Index des POST-Feldes der Eingabedaten wird die entsprechende Entität der ES gesucht und je nach Typ die passende Regel gestartet. Dabei empfiehlt es sich für jeden Typ zwei Regeln zu definieren: eine für die Überprüfung auf unerlaubte Zeichen ohne Verbindung zur Datenbank und eine zweite, die die Eingabedaten mit der Datenbank vergleicht.

Im Falle der Eingabedaten für die Kundennummer würde hier geprüft werden, ob die Spalte mit Namen "Kundennummer" vorhanden ist und ob ein Eintrag mit dem Wert "DE1" existiert.

Für die Eingabedaten des "Testproduktes" müsste die Spalte "Testprodukt" vorhanden sein und auf den Wert 0 geprüft werden. Falls hier schon ein Eintrag gemacht wurde, also schon eine Bestellung vorliegt, kann entweder die alte Bestellung überschrieben werden, die neue Anzahl aufaddiert werden, oder eine explizite Nachfrage an den Nutzer gestellt werden.

Ergibt keine Regelanwendung ein Problem, so werden die Daten aus der Eingabemaske in der Datenbank gespeichert.

Liefert eine Regelanwendung eine Rückgabe die einem Fehler entspricht, ist es sinnvoll, das weitere Vorgehen vom Rückgabetyp (falls weitere Rückgabetypen außer "true" und "false" vereinbart wurden) und der Art der Regel abhängig zu machen.

Bei fehlerhaften Eingaben in einem oder mehreren Eingabefeldern der Eingabemaske kann diese einfach nochmals aufgerufen werden und ein kurzer Hinweis für die betreffenden Textfelder eingeblendet werden.

Falls der Fehler schwerwiegender ist, z.B. bei Regeln die den Verbindungsaufbau zur Datenbank betreffen oder bei der Prüfung der ES muss das Programm abgebrochen werden und eine entsprechende Fehlermeldung (je nach Regel die verletzt wurde) ausgegeben werden.

d) Datenbank

In der Datenbank selbst gibt es noch eine Reihe von Möglichkeiten weitere Regeln anzuwenden. Beispielsweise können die Daten auf mehrfache gleiche Datensätze geprüft werden. Diese Regeln werden in dieser Arbeit allerdings nicht weiter besprochen.

Abgesehen von der Prüfung vorhandener Daten empfiehlt es sich aber für die Datenbank eine Regelgruppe bzw. ein Vorgehen zur Erstellung der Datenbank zu entwickeln, das die Tabellen der Datenbank automatisch aus der ES ableitet und erzeugt und evtl. auch eine vorhandene Datenbank mit der ES vergleicht um fehlende, überflüssige oder falsch benannte Spalten zu finden.

Ein grober Ansatz für die Erzeugung einer neuen Datenbank und den nötigen Tabellen wäre folgender:

1. Voraussetzung ist eine Datenbank in der die neuen Tabellen angelegt werden können.
2. Eine neue Tabelle in der Datenbank wird erstellt.
3. Die ES wird in einer Schleife durchlaufen und für jede Entität:
    1. Der Typ und die ID ermittelt.
    2. Eine Spalte mit dem Namen der dem Wert der ID entspricht angelegt.
    3. Der Typ der Spalte festgelegt (int, varchar etc).
    3. Je nach Typ der Ausgangswert der Spalte gesetzt (z.b. 0 für Produkte).

So wäre eine Datenbank erzeugt worden, die dem Aufbau der ES entspricht und in der sich alle Daten der Eingabemaske (die aus der gleichen ES erzeugt wird) speichern lassen. Die Namensgebung der Tabelle und der Spalten sowie der Datentypen ergibt sich aus der ES bzw. wird im Programmcode festgelegt.

IV. Regeln

a) Allgemeines Konzept für Regeln

In diesem Kapitel werden die verschiedenen Regeln beschrieben, die im Programm angewendet werden müssen um die Funktionalität des Konzepts sicherzustellen, die Eingabedaten zu überprüfen und deren korrekte Speicherung in der Datenbank zu gewährleisten. Für die Regeln bestehen verschiedene Möglichkeiten der Organisation und Verwaltung:

Konzepte für die Organisation und Verwaltung der Regeln:

1. Pro Typ eine Regel: Für jeden Entitätentyp existiert eine (große) Regel die alle notwendigen Überprüfungen vornimmt. Diese Form der Organisation ist für das Konzept "Automatische Anwendung der Regeln" (Seite 7) geeignet.

Vorteile:

- übersichtliches Regelwerk, da nur eine Regel pro Typ existiert

Nachteile:

- unflexibel
- Evtl. viel "overhead" in den Regeln, da bei ähnlichen Typen oft viele gleiche Überprüfungen anfallen.
- bei vielen Typen ein großes Regelwerk, auch wenn oftmals nur die gleichen Abfragen stattfinden.

2. Einzelne unabhängige Regeln: Ein Regelwerk mit vielen voneinander unabhängigen Regeln, die z.B. eine Eingabe auf ein oder mehrere angegebene Sonderzeichen überprüfen. Durch entsprechende Kombination dieser Regeln können komplexere Überprüfungen erreicht werden, ohne jedoch unnötigen "overhead" zu erzeugen. Diese Methode ist für das Konzept "Manuelle Anwendung der Regeln" (Seite 6) geeignet.

Vorteile:

- kaum Redundanz
- viele Kombinationsmöglichkeiten

Nachteile:

- unübersichtliches Regelwerk, da es eine Vielzahl von kleinen Regeln gibt

3. Einige große "Regelklassen", kombiniert mit einzelnen kleineren Regeln. Es wird praktisch das Regelwerk aus 2. verwendet, aber um einige Regeln/Regelklassen ergänzt, die ihrerseits aber nur mehrere der schon vorhandenen Regeln aufrufen. Die einzelnen Regeln lassen sich auch ohne diese Regelklassen direkt anwenden. Dadurch werden die ersten beiden Organisationsformen erreicht, allerdings ohne dass in den Regelklassen unnötigerweise die verwendeten Regeln doppelt implementiert werden.

Entspricht dem Konzept "Automatische und manuelle Anwendung der Regeln" (Seite 10).

Rückgabewerte:

Für die Rückgabewerte der Regeln würde sich auf den ersten Blick natürlich der bool-Typ empfehlen. D.h. es wird "false" zurückgegeben, wenn die Regel nicht eingehalten wurde, ansonsten "true". Allerdings kann es auch sinnvoll sein, weitere Rückgabewerte zu ermöglichen. So könnte beispielsweise bei größeren Regelklassen wie denen in Organisationsform 1 (s.o.) genauere Informationen zum aufgetretenen Fehler mitgeliefert werden und so die Fehlerbehandlung vereinfacht werden. Es wäre auch möglich anhand der Regelgruppe die weiteren Maßnahmen zu bestimmen. Wird z.B. eine Regel aus d) verletzt, dann könnte direkt wieder die leere Eingabemaske aufrufen werden.

In den folgenden Abschnitten werden die Regeln anhand ihrer Funktion gruppiert und beschrieben. Abgesehen von den folgenden Regeln sollten in einer fertigen Implementation noch eine weitere Gruppe von Regeln angelegt werden, die Operationen innerhalb der Datenbank ausführen, wie das Auffinden doppelter Datensätze, oder die Sicherstellung, dass Schlüsselwerte einzigartig bleiben. Auf diese Gruppe wird in dieser Arbeit nicht näher eingegangen.

Für den korrekten Programmablauf ist die Einhaltung einer bestimmten Reihenfolge in den Regelaufrufen wichtig. Als erstes müssen die Regeln aus Abschnitt b) ausgeführt werden. Tritt ein Fehler bei einer dieser Regeln auf muss das Programm abgebrochen werden, da dies bedeutet, dass die ES fehlerhaft ist und somit nicht verwendet werden kann. Danach werden die Regeln aus Abschnitt c) ausgeführt, die für die Erzeugung der Eingabemaske notwendig sind. Auch hier wird im Fehlerfall abgebrochen. Mit den Regeln aus d) werden anschließend die Eingabedaten des Nutzers geprüft. Im Fehlerfall kann ein entsprechender Hinweis auf die fehlerhafte Eingabe angezeigt werden und die Eingabemaske wird wieder angezeigt. Im letzten Schritt werden die Regeln aus e) angewendet. Hier ist die weitere Vorgehensweise nach dem Auftreten eines Fehlers abhängig von der jeweiligen Regel. Ist beispielsweise die Datenbank fehlerhaft, muss abgebrochen werden, bei einer noch nicht eingetragenen Kundennummer reicht es, dor dem Speichern der Eingabedaten ein Registrierungsformular einzublenden.

b) Regeln für die Entitätsstruktur

Die erste Gruppe enthält alle Regeln, die auf die Entitätsstruktur angewendet werden. Hier wird z.B. geprüft ob das XML-Dokument (vorausgesetzt es wird XML verwendet) vorhanden ist und den XML-Spezifikationen entspricht (d.h. ob es wohlgeformt ist). Nach diesem Test auf syntaktische Korrektheit, kann hier außerdem noch eine Validierung mittels einer Schema-Definition wie z.B. Document-Type-Defnition (DTD) vorgenommen werden. So kann die Entitätsstruktur zumindest ansatzweise auf inhaltliche Korrektheit

geprüft werden. Die Abarbeitung dieser Regelgruppe entspricht praktisch dem ersten Schritt im Ablauf des Konzepts. Die Regeln für die Entitätsstruktur stellen sicher, dass die Entitätsstruktur korrekt aufgebaut ist und allen Vorgaben entspricht, so dass sie ohne Probleme vom Hauptprogramm verarbeitet werden kann. Dadurch ergeben sich zwei Besonderheiten für diese Regelgruppe. Erstens müssen alle dieser Regeln ausgeführt werden und zweitens ist hier auch eine bestimmte Reihenfolge sinnvoll. Im Folgenden werden die Regeln für die Entitätsstruktur in ihrer empfohlenen Reihenfolge aufgeführt:

- Überprüfung, ob die Entitätsstruktur und die Strukturdefinition (falls diese als separate Datei vorliegt) existieren.

- Überprüfung der Entitätsstruktur auf syntaktische Korrektheit, damit sie vom Programm eingelesen werden kann.

- Validierung der ES mit Hilfe der SD. D.h. Es wird geprüft ob die ES nur solche Entitäten enthält, die auch durch die SD vorgesehen bzw. festgelegt sind. Falls diese Regel zum Einsatz kommt, kann die vorherige meist auch übersprungen werden, da z.b. bei einer Validierung einer XML-Datei mittels einer DTD auch gleichzeitig die syntaktische Korrektheit geprüft wird.

- Überprüfung auf Entitäten mit gleichem Namen.

Konzeptionell können diese Regeln (besonders die ersten beiden) auch eher als ein Teil des Hauptprogramms angesehen werden, da sie in der Implementierung kaum extra im Regelwerk festgelegt werden würden.

c) Regeln für die Erzeugung der Eingabemaske aus der Entitätsstruktur

Hier werden alle Regeln, die für die Erzeugung der Eingabemaske aus der Entitätsstruktur nötig sind aufgeführt. Bei der automatischen Erzeugung der Eingabemaske kommt es vor allem darauf an, eine eindeutige Verbindung zwischen den Entitäten aus der ES und ihren entsprechenden Eingabefeldern in der Eingabemaske zu halten.

- Die erste Regel ist dafür zuständig, die ID-Attribute der Entitäten in der ES durchgehen und automatisch zu vergeben, um sicherzustellen, dass alle IDs einzigartig sind. Diese IDs sind für den gesamten Programmablauf wichtig, da sie es ermöglichen, alle Entitäten gezielt anzusprechen und zu unterscheiden.

- Eine weitere Regel stellt sicher, dass jedes Objekt aus der Eingabemaske (d.h. Textfelder, Dropdown-menues, Checkboxes etc.) bei seiner Generierung seine entsprechende ID aus der ES erhält. Es wird also in einer Schleife die ES durchlaufen und für jede Entität ein dem Entitäts-Typ entsprechendes Eingabefeld erstellt. Dabei fragt die Regel in der ES den Wert des ID-Attributes der aktuellen Entität ab und setzt z.B. das "name"-Attribut eines HTML-Input-Objektes entsprechend.

d) Regeln für die Eingabedaten

Die Regeln dieser Gruppe prüfen die Eingabedaten des Nutzers auf unerlaubte Zeichen, ungültige Zeichenfolgen (z.B. für mail-Adressen) und leere Eingaben. Sie werden daher nur in Verbindung mit der schon erstellten Eingabemaske verwendet und erst dann ausgeführt, wenn der Nutzer seine Eingaben "absenden" will. Zu diesem Zeitpunkt ist keine Verbindung zur Datenbank vorgesehen bzw. nötig. Daher beschränken sich die Parameter für diese Regelgruppe auf den Inhalt des jeweiligen Eingabefeldes und des Feldtyps. Das Ziel dieser Regelgruppe sollte es sein, möglichst viele Fehler abzufangen, damit diese nicht in die Datenbank weitergeleitet werden.

Ein Beispiel für eine solche Regel wäre die Überprüfung der Kundennummer. Angenommen gültige Kundennummern fangen mit zwei Buchstaben als Länderkennung an, gefolgt von der eigentlichen Nummer, dann wären z.B. "DE123" oder "US456" mögliche Kundennummern für einen Kunden aus Deutschland bzw. den USA.

Die Regel zur Prüfung von Kundennummern bekommt nun die Eingabedaten aus dem Textfeld der Eingabemaske als Parameter (z.B. "DE007") und prüft nun, ob die Eingabe nicht leer ist, mit "DE" oder "US" (oder weiteren möglichen Kennungen) anfängt und von einer ganzen Zahl abgeschlossen wird. Es kann und wird in diesem Schritt nicht geprüft werden, ob die Kundennummer überhaupt vergeben wurde, nur ob sie den "Richtlinien" für Kundennummern entspricht.

e) Regeln zur Abgleichung der Eingabedaten mit der Datenbank

Mit den Regeln aus der zweiten Gruppe werden die Eingabedaten aus der Eingabemaske mit evtl. vorhandenen Daten in der Datenbank verglichen. So kann u.a. sichergestellt werden, dass keine schon vorhandenen Datensätze überschrieben werden, falls ein Kunde mehrmals hintereinander Produkte bestellt und nicht klar ist, ob die ersten noch nicht ausgeführten Bestellungen überschrieben werden sollen. Es kann festgestellt werden, ob alle notwendigen Tabellen zur Speicherung in der Datenbank vorhanden sind, und ob z.B. eine entsprechende Kundennummer vorhanden ist, oder ob sich der Kunde evtl. erst registrieren muss.

Nachdem im Beispiel aus Abschnitt b) die Eingabe "DE007" geprüft und für zulässig befunden wurde, wird nun eine Regel angewendet, die ebenfalls "DE007" als Parameter übergeben bekommt. Diese Regel stellt nun eine Verbindung zur Datenbank her (oder nutzt eine schon bestehende Verbindung), prüft die Datenbanktabelle auf die Existenz einer Spalte "Kundennummer" und sucht in dieser Spalte nach einem Eintrag mit Wert "DE007". So lässt sich prüfen, ob der Kunde schon registriert ist. Falls nicht, könnte eine Aufforderung zur Registrierung eingeblendet werden, oder direkt ein neuer Datenbankeintrag mit der Kundennummer angelegt werden.

V. Prototyp und Beispiel

In diesem Kapitel wird eine Umsetzung des Konzeptes mit Hilfe eines kleinen Beispiels beschrieben. Geschrieben und getestet wurde diese Programm auf Windows XP SP3, [PHP] Version 5.3.8, [MySQL] 5.5.16 und Firefox 12.0. Für die Installation und Konfiguration wurde [XAMPP] Version 1.7.7 verwendet. Um nachfolgenden Code vollständig und fehlerfrei ausführen zu können, müssen evtl. in der Datei "handler.php" die Verbindungsparameter in den Zeilen:

```
$con = mysql_connect("localhost","root","");
```

und

```
$db_selected = mysql_select_db("test", $con);
```

angepasst werden, sowie eine Tabelle "bestellung" angelegt werden, die die Spalten "kundennummer" (varchar)und "testprodukt" (int) enthält. Ausserdem wird die Datei "es.xml" durch "regelwerk.php" verändert.

Wird ein Datensatz mit der Kundennummer "DE1" angelegt, so wird das Programm bei Korrekter Eingabe der Daten den Wert für "Testprodukt" in der Datenbank speichern (und vorhandene Werte ohne Nachfrage überschreiben), ansonsten wird eine Meldung wegen der nicht vorhandenen Kundennummer ausgegeben.

Die Funktionsweise des PHP-Codes sollte sich aus den Kommentaren und der Beschreibung aus Kapitel III. Ergeben.

Dieses Programm dient nur als kurzes Beispiel, es werden nicht alle möglichen Fehler abgefangen bzw. sinnvoll behandelt!

Als Entitäten ist hier eine Kundennummer und ein Produkt vorgesehen. Gültige Eingaben für die Kundennummer müssen die Form "DEXXX" haben, wobei "XXX" eine positive ganze Zahl sein muss. Eingabewerte für das Testprodukt müssen eine positive ganze Zahl sein.

Die ES ist in diesem Fall die Datei "es.xml" mit folgendem Inhalt:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE entstruc SYSTEM "entdef.dtd">

<entstruc>

  <kundennummer id="ID1"/>

  <produkt id="ID2" name="testprodukt"/>

</entstruc>
```

Die SD als DTD ("entdef.dtd") sieht wie folgt aus:

```
<!ELEMENT entstruc (kundennummer, produkt)>
<!ELEMENT kundennummer (#PCDATA)>
<!ELEMENT produkt (#PCDATA)*>
<!ATTLIST produkt
name     CDATA      #REQUIRED
id       ID      #REQUIRED >
<!ATTLIST kundennummer
id       ID       #REQUIRED >
```

Das Programm wird in "test.php" gestartet:

```
<?php
// warnungen unterdrücken
error_reporting(0);
require('regelwerk.php');
echo "Studienarbeit Florian Lechner \n<br/><br/>";
echo "Starte Programm:<br/>";

// globale session, um Variablen auch in handler.php zu verwenden
SESSION_START();
//ein Objekt der Klasse Regelwerk erstellen.
$RW = new Regelwerk();
$_SESSION["RW"] = $RW;
// erste Regel starten und prüfen ob ES vorhanden ist
$_SESSION["RW"]->RegelGrp1R1();

//ES laden, dient als Regel zur Überprüfung der Syntax
//DOM API macht Validierung einfach
$dom = new DOMDocument();
```

```
if($dom->load("es.xml"))
  {
    echo "OK: Entitaetsstruktur laden<br/>";
   }
else
     {
       echo "Fehler: Entitaetsstruktur nicht geladen<br/>";
       echo "Programm wird abgebrochen.<br/>";
       exit;
     }

// Validierung mit der DTD
$_SESSION["RW"]->RegelGrp1R2($dom);
//ES nochmals laden mit simplexml (einfacher und praktischer als DOM)
$xml = simplexml_load_file("es.xml");
// REgel zum automatischen setzen der IDs in der ES
$_SESSION["RW"]->RegelGrp1R3($xml);

// Einagbemaske erstellen (als HTML-Form)
// handler.php wird ausgeführt, wenn der "absenden"-Button gedrückt
wird
echo "Erstelle Eingabemaske...<br/>";
?>
<form method="post" action="handler.php">
<?php
//jede Entität durchlaufen und entsprechende input-form erstellen
foreach($xml->children() as $child)
```

```
  {
   switch($child->getName())
   {
     case "kundennummer":  //falls Entität eine Kundennummer ist
     echo "Kundennummer: ";  //Namen ausgeben
     $tmp = $child->attributes()->id;
     print ("<INPUT TYPE=text name=$tmp >"); //setze name-Attribut auf
                                             //die ID der Entität
     echo "<br/>";
     break;

     case "produkt":  //falls Entität ein Produkt ist
     echo  $child->attributes()->name . ": ";  //Namen ausgeben
     $tmp = $child->attributes()->id;
     print ("<INPUT TYPE=text name=$tmp >");
     echo "<br/>";
     break;

   }
}
?>
<input type="submit" value="absenden" />
</form>
<?php
?>
```

Die Datei „handler.php“:

```
<?php
require('regelwerk.php');
```

```
SESSION_START();


// Falls keine Eingabedaten übermittelt wurden
if(!isset($_POST) OR empty($_POST))
{
    echo "Es wurden keine Daten übermittelt!";
    echo "Programm wird abgebrochen.<br/>";
    exit;
}
else
{
$xml = simplexml_load_file("es.xml");



// Alle IDs durchlaufen und für jede ID die passende Entität suchen
// dann die entsprechende Regel aufrufen
echo "Ueberpruefung der Eingabedaten wird gestartet:<br/> ";


for ($i=1; $i<=$_SESSION["maxids"];$i++)
{
  $tempid = "ID".$i;
  $result = $xml->xpath("//*[@id='$tempid']"); //suche Entität mit dem
                                                //ID-Attribut $tempid
  echo "Regeln fuer Entitaet: ".$result[0]->getName(). " mit ID: ".
$result[0]->attributes()->id." werden gestartet: <br/>";


 if($result[0]->getName() == "kundennummer")
```

```
  {
        if(!$_SESSION["RW"]->RegelTypKundennummer($_POST[$tempid]))
        {
         echo "Eingabedaten fehlerhaft!<br/>";
         echo "Programm wird abgebrochen.<br/>";
         exit;
        }
        else{ echo "Eingabedaten OK! (Inhalt: ". $_POST[$tempid].")<br/>";}
  }

 if($result[0]->getName() == "produkt")
  {
        if(!$_SESSION["RW"]->RegelTypProdukt($_POST[$tempid]))
        {
         echo "Eingabedaten fehlerhaft!<br/>";
         echo "Programm wird abgebrochen.<br/>";
         exit;
        }
        else{ echo "Eingabedaten OK! (Inhalt: ". $_POST[$tempid].")<br/>";}
  }

}
}
echo "Ueberpruefung der Eingabedaten abgeschlossen.<br/> ";
echo "Eingabedaten mit Inhalt der Datenbank prüfen:<br/> ";
```

```
//Verbindung zur DB herstellen
$con = mysql_connect("localhost","root","");
if (!$con)
  {
   die('Fehler bei Verbindungsaufbau: ' . mysql_error());
  }
else {echo "Verbindung zur Datenbank hergestellt.<br/>";}


//Datenbank auswählen
$db_selected = mysql_select_db("test", $con);
if (!$db_selected) {
    die ('Auswahl der Datenbank gescheitert : ' . mysql_error());
}
else {echo "Datenbank ausgewählt.<br/>";}



echo "Ueberpruefung der Datenbank-Regeln wird gestartet:<br/> ";


// IDs durchlaufen und die passende Entität suchen
// dann die entsprechende Regel aufrufen
//key=id value=eingabedaten
foreach ($_POST as $key => $value) {
$result = $xml->xpath("//*[@id='$key']"); //Entität mit ID "$key"
                                          //suchen


   if($result[0]->getName()=="kundennummer")
   {
     if(!$_SESSION["RW"]->RegelTypKundennummerDB($value))
```

```
     {
      echo "Kundennummer nicht vorhanden.<br/>";
      echo "Programm wird abgebrochen.<br/>";
      exit;
     }
     else {echo "Kundennummer vorhanden.<br/>";}
   }
}

// Alles ok, Anzahl des Produktes wird in DB gespeichert
echo "Speichern der Eingabedaten in DB.<br/>";
$x=$_POST['ID2'];
echo $x;
mysql_query("UPDATE bestellung SET testprodukt='$x' WHERE
kundennummer='DE1'");

SESSION_DESTROY();
mysql_close($con);

?>
```

Die Datei „Regelwerk.php“:

```
<?php
class Regelwerk{
// prüfen ob ES vorhanden
function RegelGrp1R1() {
   if(file_exists('Es.xml')){
      echo "OK: Entitaetsstruktur vorhanden\n<br/>";
```

```
   }
   else {
         echo "Fehler: Entitaetsstruktur nicht vorhanden\n<br/>";
         echo "Programm wird abgebrochen.<br/>";
         exit;
   }

}

// Validierung mit DTD
function RegelGrp1R2($dom) {
if ($dom->validate()) {
    echo "OK: DTD-Check\n<br/>";
}
else {
  echo "Fehler: DTD-Check<br/>";
  echo "Fehler: Programm wird abgebrochen.<br/>";
  exit;
}
}

function RegelGrp1R3($xml) {
// IDs setzen
$tmp="1";
$maxids=0;

foreach($xml->children() as $child)
```

```
  {
   $child['id']='ID'.$tmp;
   $tmp++;
   $maxids++;
  }

$_SESSION["maxids"] = $maxids;

//datei mit neuen ids speichern
$xml->asXML("es.xml");
echo "OK: ID'S in der Entitaetsstruktur gesetzt<br/>";
}

// Kundennummer auf korrektes Format prüfen
function RegelTypKundennummer($kn){
if($kn==""){echo "Fehler: Leere Eingabe<br/>";return false;}
if(!preg_match('/^DE/', $kn)){echo "Fehler: Nummer muss mit DE
anfangen!<br/>";return false;}
else {return true;}
}

// Produkt auf korrektes Format prüfen
function RegelTypProdukt($prod){
if($prod==""){echo "Fehler: Leere Eingabe<br/>";return false;}
if(!preg_match('#^[0-9]+$#i', $prod)){echo "Fehler: nur Zahlen erlaubt!
<br/>";return false;}
else {return true;}
}
```

```
// Prüfen, ob Kundennummer in der DB vorhanden ist

function RegelTypKundennummerDB($value){

$result = mysql_query("SELECT kundennummer FROM bestellung WHERE
kundennummer = '$value'");

$num_rows = mysql_num_rows($result);

if($num_rows==1){return true;}

else {return false;}

}

}

?>
```

In der Datei „test.php“ befindet sich der Code der für den Start des Programms zuständig ist, bis hin zum Erzeugen der Eingabemaske. Alle Programmabläufe die nach dem Klicken auf den „Absenden“-Button ausgeführt werden, befinden sich in „handler.php“. Der Quellcode für die Regeln ist in „Regelwerk.php“ gespeichert.

Um dem Programm eine neue Regel (mit Namen „RegelNeu“) hinzuzufügen und zu verwenden, muss diese als erstes implementiert werden. Dies geschieht in der Klasse „Regelwerk“ in „Regelwerk.php“. Dort muss nun eine neue Funktion mit der gewünschten Funktionalität geschrieben werden. Danach kann diese neue Regel ab der Zeile

```
$_SESSION["RW"] = $RW;
```

in „test.php“ verwendet werden, indem sie mit

```
$_SESSION["RW"]->RegelNeu();
```

aufgerufen wird.

Für den Einstieg in XML und PHP empfehle ich die folgenden Bücher:

[XML-Einstieg für Anspruchsvolle] und [PHP von Kopf bis Fuß]

VI. Fazit

In dieser Arbeit wurde ein Konzept zur dynamischen Generierung einer Eingabemaske auf Grundlage einer strukturierten Darstellung der darzustellenden Entitäten erarbeitet. In Kapitel II. wurde das Konzept aus einer nicht technologiegebundenen konzeptioneller

Sicht beschrieben. Danach wurde in Kapitel III. Auf die einzelnen Objekte des Konzepts wie die Entitätsstruktur oder die Strukturdefinition wurde in Kapitel III. näher eingegangen und das Konzept aus technischer Sicht beschrieben. Hier wurde XML, PHP und DTD verwendet. Für diese Arbeit sind diese Technologien völlig ausreichend. Als Alternative kämen für die SD u.a. XML-Schema-Definition (XSD) in Frage. An Stelle von PHP lässt sich prinzipiell jede Sprache verwenden, die eine webbasierte Eingabemaske erzeugen kann. Allerdings bietet sich PHP gerade in Verbindung von XML als Technologie für die ES an, da PHP eine Vielzahl von Funktionen zur Verarbeitung von XML-Dokumenten bereitstellt. Auf die Datenbank-Regeln und die Datenbanktechnologie wurde in dieser Arbeit nur kurz eingegangen. Das hier erarbeitete Konzept stellt keine besonderen Anforderungen an die Datenbank, so dass hier prinzipiell keine Einschränkungen in Bezug auf die verwendete Art der Datenbank besteht. Insbesondere die Regeln zur Prüfung der Datensätze innerhalb der Datenbank und die automatische Erzeugung und Aktualisierung der Datenbank bieten Raum für weitere Arbeiten zu diese Thema.

VII. Literaturverzeichnis

[mysql]:

http://www.mysql.de/

[PHP]:

http://www.php.net/, besucht : 3.Mai -14. Juni

[PHP & MySQL von Kopf bis Fuß]

Beighley, Lynn/Morrison, Michael (2009): PHP & MySQL von Kopf bis Fuß,1. Aufl. O'Reilly, 2009

[xampp]:

ApacheFriends XAMPP Version 1.7.7

http://www.apachefriends.org/de/xampp-windows.html, besucht: 3.Mai -14. Juni

[XML-Einstieg für Anspruchsvolle]

Sebestyen, Thomas (2010): XML, Einstieg für Anspruchsvolle, Addison Wesley, 2010