

Design Patterns

für plattformübergreifende Programmierung

Andreas Lichtenberger

Inhaltsverzeichnis

<i>1. Allgemeines</i>	<i>3</i>
<i>2. Pattern für plattformunabhängiges programmieren</i>	<i>6</i>
<i>2.1. Datentypen</i>	<i>6</i>
<i>2.2. Unicode</i>	<i>6</i>
<i>2.3. Scheduling</i>	<i>7</i>
<i>2.4. Protokolle</i>	<i>8</i>
<i>2.5. Virtuelle Maschinen</i>	<i>9</i>
<i>2.6. Emulatoren</i>	<i>11</i>
<i>2.7. Middleware</i>	<i>12</i>
<i>2.8. Bibliotheken</i>	<i>13</i>

I. Allgemeines

Das beste Beispiel für ein plattformübergreifendes System ist das Internet. Es wurde von Anfang an so entworfen, dass jede Plattform mit dem Internet verbinden kann. Entsprechend wurden damals umsichtig universelle Protokolle für den Datenaustausch festgelegt.

OSI	TCP/IP					
Anwendung	RIP	TELNET	RLOGIN	NFS, NIS	Application	
Darstellung	HELLO	FTP	RSHELL	NICS TIME		
Sitzung	EGP	TFTP	REXEC	XDR		
	DNS	SMTP		RPC		
Transport	TCP		UDP		Transmission	
Vermittlung	IP	ICMP	ARP	RARP	sonstige	Internet
Sicherung	SLIP, X.25, IEEE 802.x, usw.				Netzwerk	
Bitübertragung						

Definition der Plattformunabhängigkeit

Plattformunabhängigkeit ist die Eigenschaft eines Programms, auf verschiedenen Computersystemen mit Unterschieden in Architektur, Prozessor, Compiler, Betriebssystem und weiteren Hilfsprogrammen, die zur Übersetzung notwendig sind, lauffähig zu sein.

Unter plattformübergreifenden Programmieren versteht man es, einen Quellcode zu schreiben, dessen Übersetzung ohne Veränderungen auf mehreren Plattformen läuft. Als ein plattformunabhängiges Programm bezeichnet man lediglich ein Programm, dass in verschiedenen Versionen für verschiedene Architekturen vorliegt. Hierbei muss der Quellcode nicht der selbe sein.

Formen von Plattformunabhängigkeit bei Programmen:

Zwischencode: Programme, die entweder in Bytecode, wie bei Java oder in einem portablen, interpretierbaren Quellcode wie bei Perl oder Python vorliegen.

Fat Binaries: Programmpakete, die mehrere lauffähige Versionen enthalten. Das Betriebssystem startet automatisch die richtige Version. Voraussetzung für das Erstellen von Fat Binaries ist die Möglichkeit, den Quellcode zu portieren.

Eingesetzt werden sie zum Beispiel unter Mac OS X für Programme, die unter PowerPC und unter x86 Plattformen laufen sollen.

Quellcode-Portabilität: Häufig bei C Programmen, die zum Beispiel unter Unix und unter Windows Systemen laufen sollen. Der Quellcode enthält Anweisungen, die es erlauben, die Betriebssystemunterschiede auszugleichen. Hierfür werden unter anderem Präprozessordirektiven wie *#ifdef* verwendet. Es gibt auch einige nützliche Hilfsmittel hierfür wie zum Beispiel GNU autoconf.

Viele im Quellcode portable Programme stehen bereits in vorgefertigten Versionen für verschiedene Plattformen bereit.

Beispiele hierfür sind: Mozilla, Gimp oder OpenOffice.

Eingeschränkte Plattformunabhängigkeit: wenn ein Programm zum Beispiel nur auf einem bestimmten Prozessortyp aber ansonsten auf verschiedenen Hardwarearchitekturen läuft. Dies ist häufig bei Assemblersprachen der Fall. Auch Programme, die unabhängig vom CPU Typ auf nur einer Betriebssystemfamilie laufen gelten als eingeschränkt Plattformunabhängig.

Möglichkeiten plattformunabhängige Applikationen zu erstellen

Anwendung in einem *plattformübergreifenden Programmiersystem* (Java, .NET) schreiben. Eignet sich für große, neu zu entwickelnde Programme, wie zum Beispiel Textverarbeitungen, Datenbank-applikationen.

Ein Cross-Plattform Framework, also eine API verwenden, die auf mehreren Plattformen zu Hause ist. Eignet sich für große Anwendungen, die auf maximale Geschwindigkeit ausgelegt sind, wie zum Beispiel Browser, Multimediaprogramme oder Spiele.



Daten in einem universellen Format speichern und Abspielprogramme für jede Plattform schreiben (Director, Flash, Browser). Geeignet für Anwendungen, die einem vorstrukturierten Schema folgen. Zum Beispiel Multimediaabspielprogramme. Es kommen einfache Interpreter und Skriptsprachen zum Einsatz (Lingo, Java Skript).

Design Patterns und Plattformunabhängigkeit

Die meisten in dieser Veranstaltung angesprochenen Pattern für die Programmierung (Singleton, Flyweight, Decorator) sind plattformübergreifend gültig.

II. Patterns für plattformunabhängiges programmieren

Datentypen Pattern

Wichtig, um plattformunabhängigen Code zu erzeugen ist es, darauf zu achten, dass darauf geachtet wird, dass die vom Programm verwendeten Variablen aus Datentypen bestehen, deren Wertebereich und Größe auf allen Zielsystemen gleich sind, beziehungsweise bei der Codeerstellung darauf geachtet wird, dass Probleme behandelt oder verhindert werden, die aufgrund unterschiedlicher Wertebereiche und Größe entstehen.

So ist zum Beispiel in der Programmiersprache C nicht klar festgelegt, wie groß die Ganzzahldatentypen short, int und long sein müssen. Es gibt Architekturen, da ist ein short kleiner als ein int, während sie auf anderen gleich groß sind.

Mit Hilfe des sizeof Operators kann in C herausgefunden werden wie viel Speicherplatz eine Variable eines bestimmten Typs auf einem System belegt.

Bei vorzeichenbehafteten Datentypen erhöht sich bei C der Wert für den negativen Zahlenbereich um eins auf Maschinen, auf denen negative Zahlen als Zweierkomplement dargestellt werden.

Die Programmiersprache Java garantiert hingegen, dass alle ihre Datentypen plattformübergreifend eingesetzt werden können.

Unicode Pattern

Für die plattformunabhängige Programmierung muss immer darauf geachtet werden, welcher Zeichensatz von der Plattform unterstützt wird. Sinnvoll ist es, Strings immer in Unicode (UTF-8) zu codieren, da es sonst bei länderspezifischen Zeichen zu schweren Darstellungsfehlern kommen kann.

Wird ein entsprechender Zeichensatz vom Zielsystem nicht unterstützt, so muss darauf geachtet werden, dass auch nur Zeichen des unterstützten Zeichensatzes verwendet werden.

Scheduling Pattern

Sollen plattformunabhängige Programme mit mehreren nebenläufigen Funktionen entwickelt werden, so muss beachtet werden, dass die unterschiedlichen Plattformen unter Umständen einen anderen Schedulingalgorithmus implementiert haben, was heißt, dass der Wechsel zwischen den nebenläufigen Programmabläufen wahrscheinlich zu einem anderen Zeitpunkt stattfindet. Dies kann zu ungewollten Programmverhalten und zu schweren Fehlern führen.

Beispiel: Ein Programm stellt seine unterschiedlichen Bildschirme mit einem Thread dar. Auf jedem der Bildschirme gibt es den Knopf "ok". Die Ereignisverarbeitung für diesen Knopf findet in einem anderen Thread statt. Je nachdem in welchem Bildschirm sich der erste Thread befindet, für das betätigen des Knopfes eine unterschiedliche Aktion ausgeführt.

Die richtige Aktion wird durch eine Switch/Case Kette bestimmt. Nehmen wir an, eine Plattform wechselt nach betätigen des Knopfes in den Thread des Eventhandlers, führt die Aktion aus und bleibt in diesem Thread, bis der Eventhandler komplett durchlaufen ist bevor sie wieder in den Thread für die Darstellung wechselt. Die Applikation läuft in diesem Szenario Fehlerfrei.

Eine andere Plattform führt im Eventhandler nur die Aktion aus und wechselt sofort zurück in den Thread für die Darstellung. Dort wird der Bildschirm gewechselt und im Anschluss wieder in den Thread des Eventhandlers gegangen. Dieser führt jetzt den Handler an der Stelle weiter aus, in dem er zuvor Unterbrochen wurde. Der Handler reagiert jetzt, als wäre der Knopf "ok" schon gedrückt worden obwohl dies gar nicht der Fall ist. Es wird eine nicht gewollte Aktion durchgeführt, die zu einem Falschen Verhalten führt.

In diesem Beispiel könnte man den Handler nach einer ausgeführten Aktion durch ein Return verlassen oder den gesamten Handler synchronisieren.

Allgemein gilt es, die unterschiedlichen Schedulingverhalten durch synchronisieren oder andere geeignete Maßnahmen zu kompensieren, ohne den Funktionen zu viel an Nebenläufigkeit zu nehmen.

Protokollpattern

Ein weiteres, häufig mit unterschiedlichen Plattformen ist es, dass sie wenn sie Daten senden oder empfangen nicht immer die gleichen Protokolle unterstützen oder die gleichen Protokolle unterschiedlich implementiert sind.

Man sollte also bei einer plattformunabhängigen Programmierung darauf achten, dass nur Protokolle verwendet werden, die von so vielen Plattformen wie möglich unterstützt werden. Oder man prüft im Vorfeld sorgfältig, welche Protokolle auf welche Weise von den Zielplattformen angeboten werden.

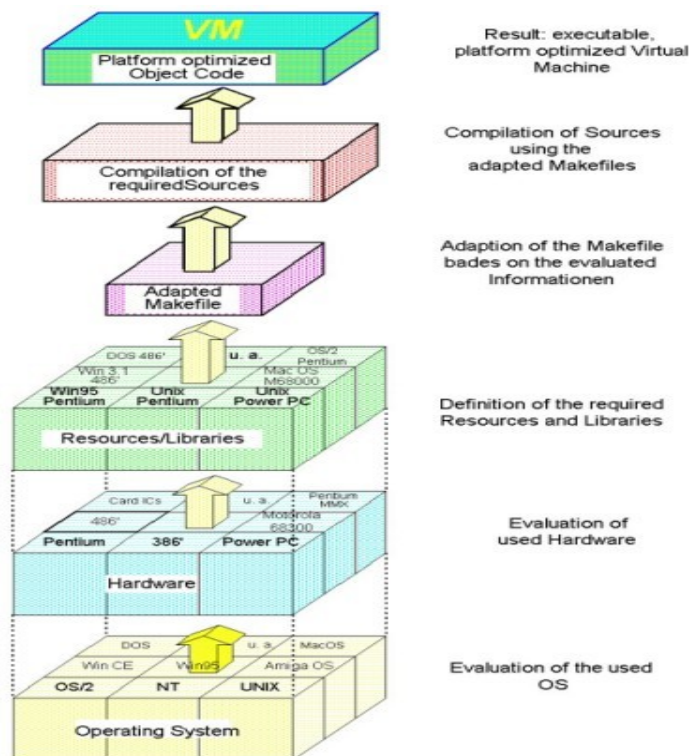
Für den Netzwerkverkehr auf unterer Ebene sollte man in diesem Zusammenhang mindestens TCP/IP und UDP nennen.

Für den Datenaustausch auf Applikationsebene muss man auf jeden Fall XML erwähnen.

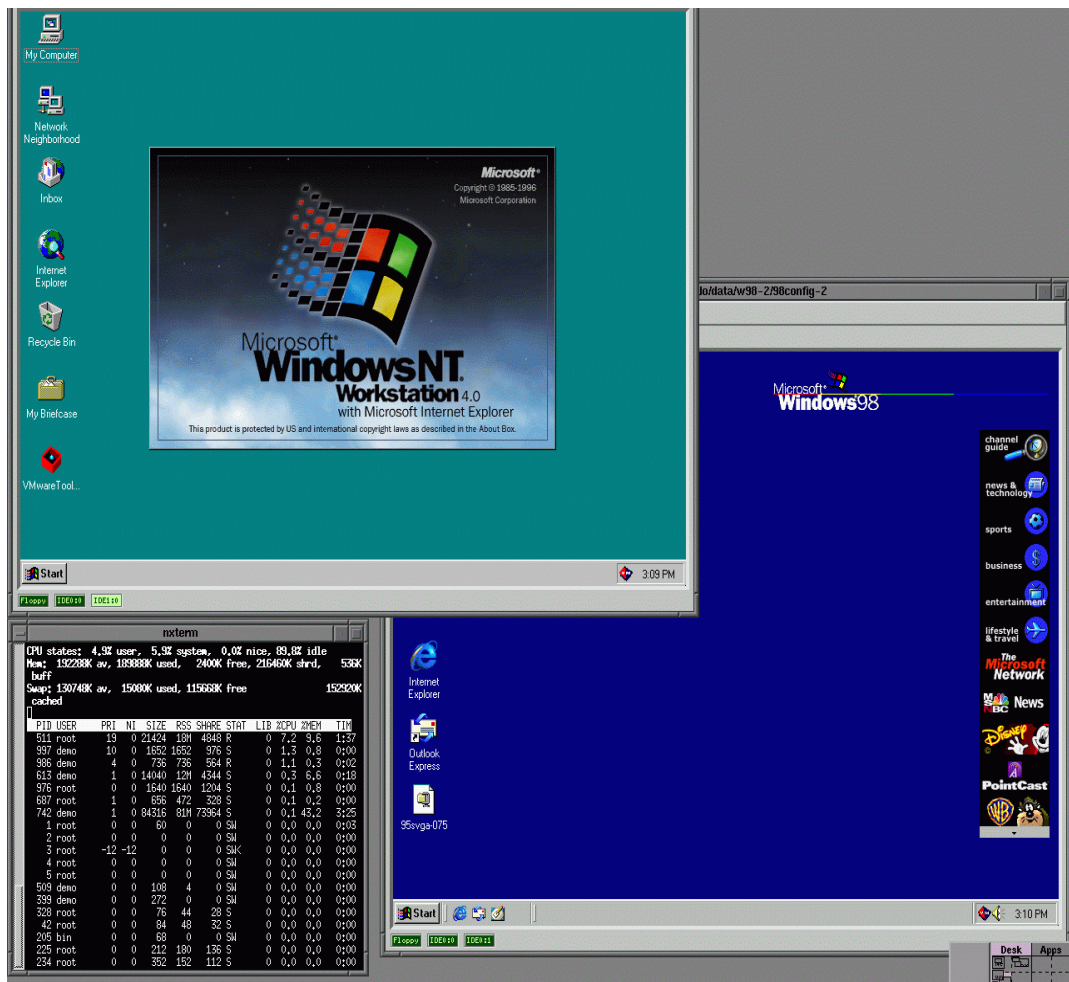
VM-Pattern

Ist es nicht möglich oder zu aufwendig plattformübergreifenden Code zu schreiben gibt es die Möglichkeit eine virtuelle Maschine einzusetzen. Wir trennen die virtuellen Maschinen in zwei grundlegende Arten.

Die eine Art stellt eine Laufzeitumgebung für in einem bestimmten Bytecode vorliegende Applikationen zur Verfügung. Sie kapselt also die Hardwarearchitektur und das Betriebssystem für die Applikation. Die Applikation wird dann für die virtuelle Maschine geschrieben und die virtuelle Maschine für die verschiedenen Plattformen bereitgestellt. Das wohl berühmteste Beispiel hierfür ist die JVM (Java Virtual Maschine)

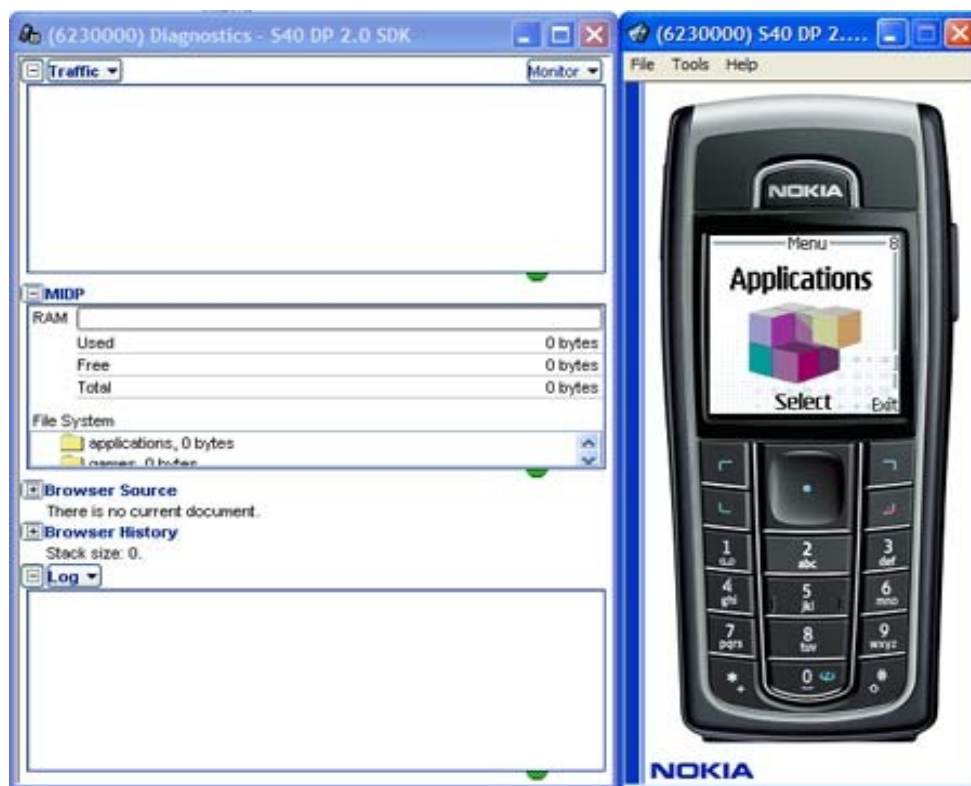


Die zweite Art stellt eine Laufzeitumgebung für ein Betriebssystem zur Verfügung. Es läuft also in der virtuellen Maschine nicht mehr nur die Applikation, sondern das gesamte Betriebssystem, für welches die Applikation geschrieben wurde und auf diesem Betriebssystem die eigentliche Applikation.



Emulatorpattern

Eine weitere Möglichkeit plattformspezifischen Code plattformübergreifend zu nutzen bieten Emulatoren. Im Gegensatz zu den virtuellen Maschinen bieten Sie nicht eine Laufzeitumgebung für eine Applikation oder ein Betriebssystem, sondern simulieren eine komplette Hardwarearchitektur samt Betriebssystem und Besonderheiten. Sie werden oft dazu genutzt, um Software für zum Beispiel mobile Endgeräte oder embedded Systems auf anderen Systemen (PC) zu nutzen oder zu testen.

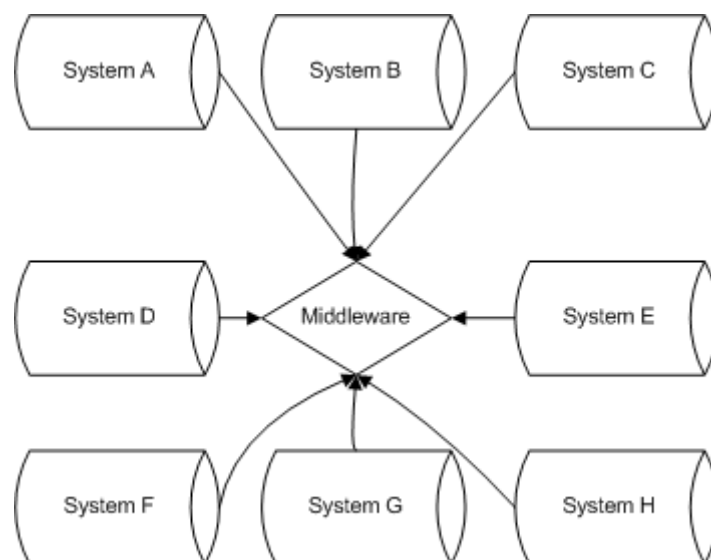


Middlewarepattern

Die dritte Möglichkeit um die Unterschiede verschiedener Plattformen auszugleichen, ohne plattformunabhängigen Code zur Verfügung zu haben ist die Middleware.

Von Middleware spricht man in vielen Bereichen. Prinzipiell überbrückt sie Ungleichheiten zwischen verschiedenen Systemen. Sie kann hierbei zwischen verschiedenen Betriebssystemen stehen, zwischen Betriebssystem und Applikation oder aber auch zwischen verschiedenen Applikationen. Sie übernimmt zum verschiedene Aufgaben. Zum Beispiel die Umsetzung von Protokollen. Sie nimmt die Daten in dem Protokollformat des einem Systems entgegen und liefert sie an das andere System in dessen Format.

Ein gutes Beispiel für eine Middleware ist ein Proxi.



Bibliothekspattern

Die meisten Programmiersprachen, in denen plattformübergreifend programmiert werden kann, greifen auf so genannte Bibliotheken (Libraries) zurück. Sie liefern Funktionalitäten, die im Kern der Sprache nicht vorhanden sind. Nicht alle Bibliotheken sind in allen Plattformen vorhanden. Auch gibt es Bibliotheken von Drittanbietern, die nicht für alle Plattformen vorhanden oder umsetzbar sind. Bei der Planung einer plattformunabhängigen Applikation muss also beachtet werden, ob alle benötigten Bibliotheken von allen Zielplattformen unterstützt werden, oder ob ein Weg besteht nicht vorhandene Bibliotheken zu ersetzen oder für die Zielplattformen umzusetzen.

Auch für plattformunabhängige Sprachen wie Java gibt es solche Bibliotheken. Als JSR (Java Spezification Request) bezeichnet spielen sie zum Beispiel bei der Applikationsentwicklung für mobile Endgeräte eine große Rolle.

3. Einführung in plattformübergreifendes HTML/CSS

Das Problem bei der Entwicklung von Websites mit Cascading Style Sheets ist bekannt. Die gängigen Browser unterstützen nur Teile des CSS Standards und halten sich oft nicht an die Definitionen, also interpretieren Angaben falsch.

Wie ist es möglich, eine Website mit CSS zu gestalten, so dass sie in allen Browsern auf die **selbe** Weise dargestellt wird und **keine** Darstellungsfehler enthält?

4. Patterns für HTML / CSS

4.1. Conditional Comments

Conditional Comments erlauben es speziellen HTML-Code für den Microsoft Internet Explorer in seine Website zu integrieren. Diese Kommentare funktionieren ab Version 5 und erlauben sogar das Erkennen unterschiedlicher Versionen (5.0,5.5,6,7).

Beispiel:

```
<!--[if IE]>  
  <style>  
    spezielle Eigenschaften für den IE  
  </style>  
<![endif]-->
```

Conditional Comments werden nur bedingt eingesetzt, da es sich ja nur speziell um den IE handelt. Das empfiehlt sich nur dann, wenn eine Website bereits auf gängige Browser abgestimmt ist und nur noch der IE einer Anpassung bedarf. Was nicht selten der Fall ist, da der IE im Gegensatz zu seinen Kollegen Firefox, Opera und Konqueror CSS nur mangelhaft unterstützt.

4.2. Global Reset

Ein anderes Problem stellen die vordefinierten HTML-Tags dar. Wie z.B. <h1>, <h2>, <p> usw., diese werden von jedem Browser mit unterschiedlichen Eigenschaften versehen (Größe, Auszeichnung und Abständen). Deshalb empfiehlt es sich, um wirklich ein gleiches Bild in allen Browser zu bekommen diese Eigenschaften zu Beginn zurückzusetzen.

Den Reset erreicht man durch:

```
* {  
    margin: 0px;  
    padding: 0px;  
    border: 0;  
}
```

Im CSS-File oder Style-Tag. Nun sitzen alle Elemente direkt aufeinander, aber in allen Browsern. Nun muss man nur noch die gewünschten Abstände für die verwendeten Elemente angeben, und sie sitzen, wo sie hin sollen.

4.3. Boxmodell

Das Boxmodell spart Codezeilen und schafft Übersicht im Quelltext. Doch leider hat der IE wieder Probleme mit der Umsetzung.

Ein Beispiel:

```
div {  
    width: 100px  
    margin: 15px;  
    padding: 10px;  
    border: 2px solid black;  
}
```

Dies ist eine Box mit einer Breite von 100px + den Aussenabstand von 2mal 15px einen Innenabstand von 2x10px und den Rahmen mit 2mal 2px, das macht eine Gesamtbreite von 154px und nicht wie gedacht 100px.

Der Internetexplorer addiert hier die Breite falsch:

Er ignoriert, bis Version 5.5, bei der Berechnung der Breite die Rahmen und Das Padding (Innenabstand). So kommt er auf eine Breite von 130px.

Dieser Fehler lässt sich leider nicht mit dem Globalreset vermeiden, sondern erfordert andere Maßnahmen. Entweder verzichtet man auf die Innenabstände, was meistens möglich ist, oder man verwendet für den IE eine eigene Styelist mithilfe der Conditional Comments.

4.4. Skalierbarkeit

Wie erreicht man, dass die Zoomfunktion des Browsers die Vergrößerung und Verkleinerung richtig darstellt?

Dafür muss man im den Eigenschaften fixe Werte wie pt und px vermeiden, diese skalieren nicht mit. Für Skalierbare Elemente gibt es die Einheiten % und em. Diese Beziehen sich immer auf das Elternelement und werden vererbt.

Deshalb empfiehlt es sich die Größe im obersten Element, also dem body zu initialisieren.

```
body{  
  font-size: 62.5%; /* Resets 1em to 10px */  
}
```

In diesem Fall ist 1em genau 10px, damit lässt sich leicht rechnen. So kann man einer Box zum Beispiel die Breite 800px geben, in dem man sie 80em groß macht, so skaliert sie immer mit. Bei diesem Verfahren, muss man allerdings wegen der Vererbung stark aufpassen, Schriftgrößen sollten immer für das niedrigste Element in der Vererbung gesetzt werden da man sonst in Verschachtelungen wie schnell den Überblick verliert und unerwartete Ergebnisse bekommt.

5. Fazit für HTML / CSS

CSS ist eine gute Möglichkeit Websites zu gestalten. Leider lässt die Kompatibilität in manchen Browser sehr zu wünschen übrig aber mit diesen Ansätzen lässt sich das meiste umsetzen. Bleibt nur zu Hoffen, dass die Browserhersteller die nächsten Version besser umsetzen. Der Internet Explorer 7 hat seinen guten Willen schon gezeigt – leider nicht mehr.