# Editorial

**Liebe Leserinnen und liebe Leser,**

Sie finden in dieser Ausgabe die Proceedings des 25. Workshops Software-Reengineering und -Evolution der Fachgruppe Software Reengineering (SRE), einen Artikel über den Car DevOps Approach, den Aufruf zu Beiträgen für das Sommertreffen der Fachgruppe Requirements Engineering zum Schwerpunktthema Requirements Engineering für das Internet of Trusted Things (IoTT) und die Ankündigung des REFrame-Workshops auf der International Requirements Engineering Conference in Hannover, ebenfalls aus der RE-Fachgruppe. Ich freue mich, dass unsere Softwaretechnik-Community so eifrig neue Erkenntnisse und Ideen erzeugt!

*Andrea Herrmann*

# Adapter-Muster

Früher, als die guten alten Zeiten schon vorbei waren, aber die Digitalisierung uns noch nicht das Leben erleichterte, in einer dunklen Zwischenphase des technologischen Chaos, da kam es vor, dass man ins Ausland reiste und dort einiges nicht passte: fremdes Geld, seltsame Steckdosen, ungewohnte Türklinken. Geld ließ sich eintauschen, an Klinken sich gewöhnen, aber wenn man mit dem falschen Stecker ankam, dann funktioniere der Föhn nicht. Mehr Elektrogeräte hatte frau damals noch nicht im Gepäck.

Tja, aber was tat man dann? Das Herumschrauben an den Steckdosen war natürlich implizit verboten. Man wollte sich aber auch nicht für jedes Land einen passenden Föhn zulegen. Für diesen Zweck wurden der Adapter erfunden. So konnte man einen deutschen Föhn an einer Commonwealth-Steckdose betreiben oder auch umgekehrt. (Natürlich benötigte man für jeden der beiden Fälle einen anderen Adapter.)

Das Problems resultierte daraus, dass sich in verschiedenen Ländern unterschiedliche Standards für die Steckdosen und Stecker durchgesetzt hatten, die zwar landesweit galten, aber selten darüber hinaus. Im Zuge der Globalisierung von Dienst- und Privatreisen wurden diese lästigen Unebenheitenn weggebügelt. Heute kann man seinen deuschen Föhn in der ganzen EU betreiben.

Solche Kompatibilitätsprobleme treten natürlich nicht nur bei Elektrogeräten auf, sondern auch auf der atomaren Ebene zwischen Code-Klassen. Die Klassen Foehn und Steckdose sollen zusammenarbeiten, aber sie passen nicht. Das heißt, die Ausgabe einer Funktion von Steckdose hat nicht die Form, die Foehn als Eingabe benötigt. Wie auch? Sie wurden in unterschiedlichen Ländern durch unterschiedliche Autoren hergestellt. Auch für dieses Problem gibt es zwei Lösungen: langfristig eine Standardisierung (wie durch den Schnittstellen-Standard WSDL für Web Services) und kurzfristig durch eine Adapter-Klasse.

Da dieses Problem häufig auftritt, wurde als wiederverwendbare, generische Lösung das Entwurfsmuster namens Adapter definiert. Sein Zweck ist es, Schnittstellen zu übersetzen, d.h. die Ausgabe von Steckdose wird in eine passende Eingabe für Foehn umgewandelt. Also wie ein Übersetzer. Der Vorteil ist, dass die Klassen selbst nicht verändert werden müssen. Man kann eine handelsübliche Foehn-Klasse und Steckdosen-Klasse verwenden. Wir wollen und dürfen ja nicht alles überarbeiten, das wir benutzen.

Abbildung 1 stellt das Prinzip des Adapter-Musters dar: Die Klasse Klient möchte die Methode Dienst.service() nutzen, ruft jedoch stattdessen die abstrakte Methode Schnittstelle.operation() auf, die wiederum von Adapter.operation() so implementiert ist, dass diese Dienst.service() aufruft und ausführt, aber das Ergebnis in der für den Klienten richtigen Form liefert.

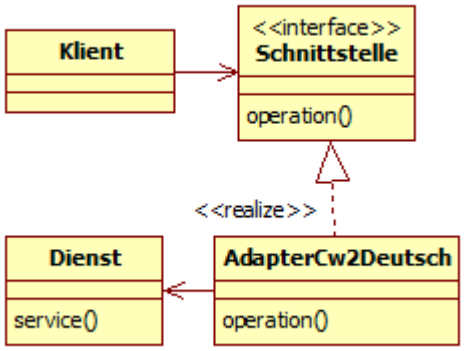


Abbildung 1: Das Adapter-Muster

In Abbildung 2 nun dasselbe nochmal konkret: Der deutsche Föhn möchte Strom nutzen, ruft die AdapterSchnittstelle auf, die wiederum vom AdapterCw2Deutsch implementiert wird. Innerhalb der Methode AdapterCw2Deutsch.stromNutzen() wird die Methode CommonwealthSteckdose.stromLiefern() aufgerufen. Problem gelöst!

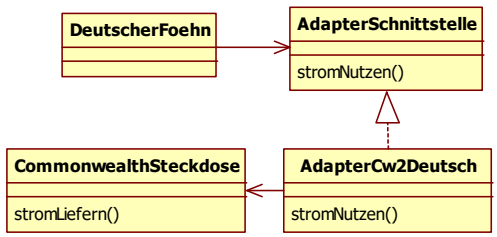


Abbildung 2: Die Klassen Foehn und Steckdose sowie ihr Adapter

Manchmal, falls es die Programmiersprache erlaubt, wird das Muster auch durch eine Doppelvererbung umgesetzt, wo der Adapter eine Tochter-Klasse sowohl von Schnittstelle als auch von Dienst ist.

Das Adapter-Muster wird auch Wrapper genannt, weil es die zu nutzende Klasse mit einer neuen Oberfläche „umhüllt“. Leider kostet diese Umwandlung Ausführungszeit. Und benötigt für jede Klient-Dienst-Kombination einen eigenen Adapter!

*Andrea Herrmann*