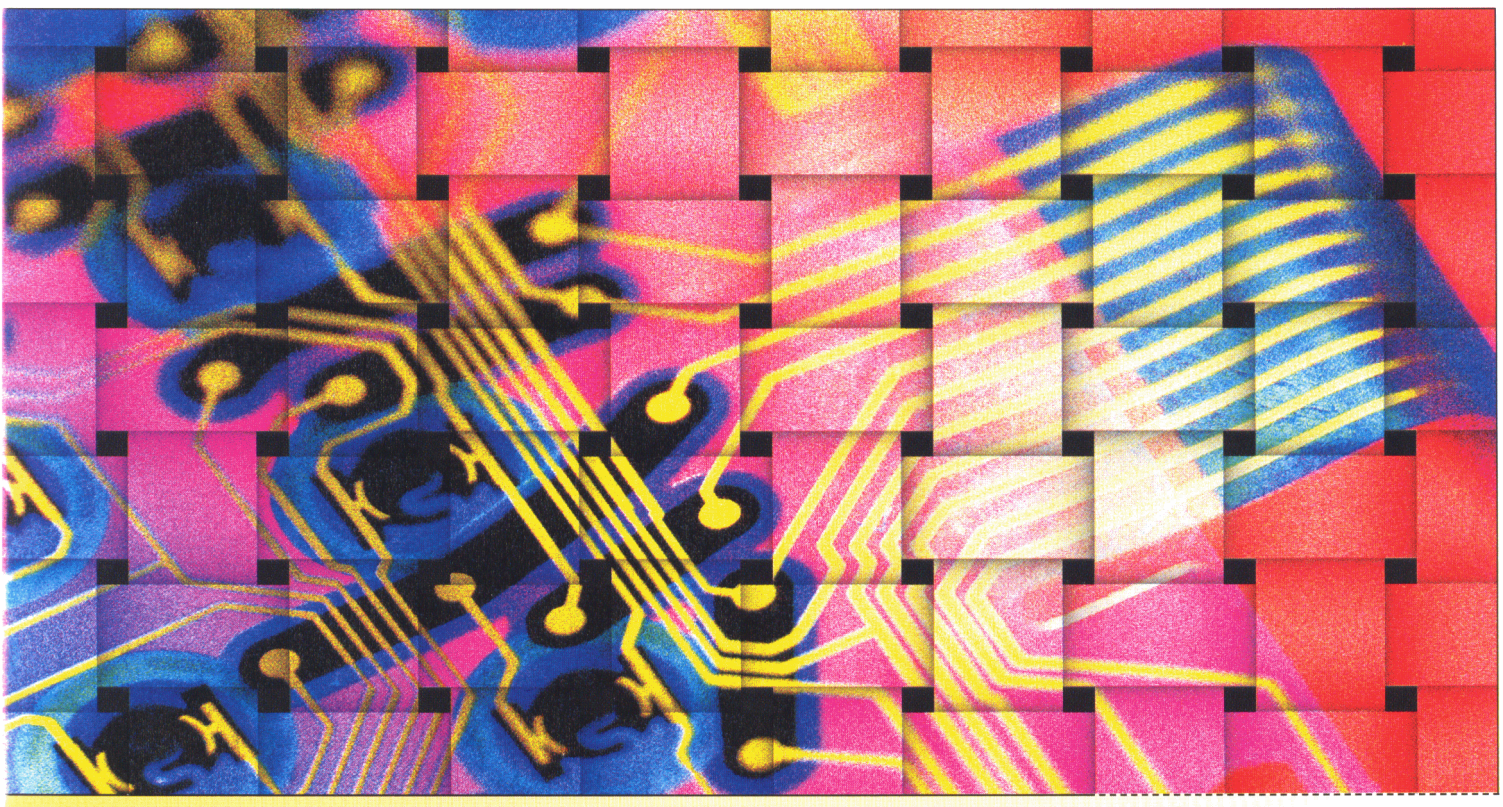
Entwurfsmuster und komponentenbasierte Softwareentwicklung

Prof. Dr. Max Mühlhäuser, Dipl.-Inform. Markus Schumacher,
Prof. Dr. Jan Borchers



Einleitung

Die Entwicklung der Komponententechnologie begann schon vor vier Jahrzehnten und verlief – vereinfacht gesagt – in folgenden fünf Schritten, die sich nicht ablösten sondern ergänzten:

1. Abstrakte Datentypen (ADT): Ein ADT ist eine Komponente, die Daten mit den direkt dazu gehörigen Operationen bündelt. Die eigentlichen Daten können von Komponenten-Nutzern nicht direkt eingesehen oder verändert werden; diese *Kapselung* und das *Verbergen (information hiding)* der Operations-Implementierungen vor den Nutzern stellen sicher, dass Daten nur in der vorgesehenen Weise benutzt werden können und Implementierungsdetails unabhängig von den Nutzern geändert werden können (Fehlerbeseitigung, effizientere Algorithmen usw.). ADTs beschreiben kein individuelles Objekt, sondern eine Objektklasse. Die Schnittstellen der Operationen (Regeln über den Aufruf, Vor- und Nachbedingungen) beschreiben abstrakt einen Objekttyp.

2. Objekte: Der Begriff Objektklasse statt ADT (bisweilen mit weniger strenger Kapselung) wurde gebräuchlich im Zusammenhang mit der Idee, *Hierarchien* von Objektklassen zu bilden, die von der universellen ‚Wurzel'-Klasse über Unterklassen (wie z.B. ‚Drucker') zu immer feineren Spezialisierungen (Duplex-Farblaserdrucker etc.) reichen. Klassenhierarchien ermöglichen bessere Modularisierung und erlauben es, ‚Fallunterscheidungen' auf das Nötigste zu beschränken. Operationen wie z.B. *GifBild-Drucken(BildDateiName)* können bei identischer Schnittstelle für verschiedene Spezialisierungen unterschiedlich realisiert werden (Polymorphie: Vielgestaltigkeit), die Wahl der erforderlichen Spezialisierung kann ggf. erst während des Programmablaufs erfolgen (dynamisches Binden).

3. Komponenten: Vergleicht man modulare Software mit auf eine Platine gelöteten ICs, so wird man wohl als wichtigste Aufgaben beim Platinenentwurf die korrekte, zielführende *Auswahl* und *Ver-*

Patterns and component based software development

Formal methods and programming language approaches alone are not sufficient for coping with software components, their customization and assembly. A concept called „software (design) patterns" has become prominent for semi formal descriptions of solutions to common problems in software engineering. The following article introduces this concept in general, and illustrates it in two application domains: software engineering for secure systems and development of interactive software.

drahtung der ICs sehen. Was die Auswahl betrifft, stimmt der Vergleich mit modularem Softwareentwurf recht gut: für ICs gibt es Herstellerkataloge, welche Ähnlichkeit mit einer Objektklassen-Hierarchie haben (Baugruppen mit diversen Spezialisierungen). Der hier beschriebene dritte Schritt – vom objektorientierten Ansatz zur Komponententechnologie im engeren Sinne – versuchte, ähnlich dem Platinenentwurf auch Zusammenspiel und ‚Verdrahtung' von Objekten besser zu unterstützen.

4. Aspekte: Die Zerteilung von Aufgaben in geschlossene Komponenten ist nicht immer möglich. Häufig gibt es so genannte *cross cutting concerns,* die sich auf viele Komponenten beziehen, wie Zugriffsrecht-Prüfung, Transaktions-Unterstützung usw. Aspektorientierte Programmierung unterstützt möglichst automatisiertes ‚Verweben' der Aspekte mit Komponenten und die Beschreibung von Aspekten möglichst wartungsfreundlich und unabhängig von den zu durchwebenden Komponenten.

5. Entwurfsmuster (Patterns): Im vorliegenden Artikel geht es um den fünften ‚großen' Schritt. Man erkannte, dass man nicht alle Abläufe und Regeln vollständig formal als wieder verwendbare Komponenten beschreiben kann. Was sich der Formalisierung am hartnäckigsten entzieht, sind Erkenntnisse, die bei erfahrenen Softwareentwicklern zu finden sind: Entwurfsmuster bezeichnen *bewährte Lösungen für wiederkehrende Probleme,* die nicht unmittelbar formal (z.B. als Algorithmen) beschrieben werden können.

Herkunft und Forschung

Geprägt und erstmals systematisiert wurde das Konzept von Entwurfsmustern (Patterns) nicht in der Softwaretechnik, sondern in der Architektur. Der berühmte Architekt und Stadtplaner Christopher Alexander stellte in den 70er Jahren systematisch Entwurfsmuster auf, die zu durchdachten stadtplanerischen Konzepten führen sollten. Dabei wurde klar, dass Patterns auf verschiedenen Detaillierungsstufen des Entwurfs vorkommen und ineinander greifen. Pattern-Forschung ist von einer Besonderheit geprägt: Während Forscher normalerweise innovative und originäre Ansätze suchen, dürfen Patterns selbst gerade dieses nicht sein, da sie sonst nicht als bewährt gelten könnten. Im Gegenteil, eine goldene Regel besagt, dass erst nach mindestens drei erfolgreichen Anwendungen eines Patterns auf hinreichend disjunkten Gebieten die Bezeichnung überhaupt zulässig ist. Pattern-Forschung konzentriert sich also nicht auf die „Erfindung" neuer Patterns, sondern auf deren Herausarbeiten, auf Pattern-Systematiken (Beschreibungskonzepte, -Sprachen etc.) sowie Rechnerunterstützung.

Beschreibung von Entwurfsmustern

Zwar sind Entwurfsmuster nicht streng formal zu beschreiben, doch darf man ihre Formulierung nicht der Beliebigkeit überlassen, wenn sie systematisch als Erfahrung weitervermittelt, bei Bedarf effizient gefunden und in ihrer Verzahnung verwendet werden sollen. Daher hat sich durchgesetzt, die Beschreibung von Patterns halbformalen Beschreibungsschemata zu unterziehen und dabei dem Beschreiber kritische Fragen bzw. Aufgaben zu stellen. Die zur vollständigen oder hinreichenden Beschreibung eines Patterns auszufüllenden ‚Felder' sind noch nicht mit breitem Konsens standardisiert, vermutlich werden sich für verschiedene Problembereiche der Softwareentwicklung leicht unterschiedliche Varianten durchsetzen. Nachfolgend werden kurz diejenigen Felder beschrieben, die sich in unserer Forschung über Entwurfsmuster für interaktive Software herauskristallisiert haben; viele davon finden sich abgewandelt auch in der Pattern-Forschung für andere Problembereiche wieder, wie der Abschnitt über Sichere Systeme zeigen wird:

Name: Auf den ersten Blick trivial, ist es in Wahrheit oft schwer, einen prägnanten, möglichst breit akzeptierten Namen für ein Entwurfsmuster zu finden

Erläuterung: Wenn nicht anzunehmen ist, dass der Leser unmittelbar das Problem und die bei seiner Beschreibung verwendeten Termini versteht, muss über den

Namen hinaus eine Hinführung in den Themenbereich erfolgen.

Ranking: Der Beschreiber sollte sein Vertrauen in ein Pattern grob abschätzen (z.B. mit ‚Sternen' ähnlich Hotel-Kategorien) um aufzuzeigen, wie erprobt und wie breit einsetzbar das Entwurfsmuster ist.

Illustration: Auch dieses Feld mag verwundern, aber wenn ein Bild mehr sagt als tausend Worte, kann ein prägnantes Foto, Comic oder eine Schemazeichnung sehr gut auf einen Blick das vom Pattern behandelte Problem illustrieren; oft werden Beschreiber von dieser Aufgabe nicht angetan sein, der Nutzen ist aber erfahrungsgemäß hoch, insbesondere wenn Pattern-Sammlungen umfangreich werden.

Kräfte/Anforderungen: Die Randbedingungen und deren Einfluss auf das Entwurfsmuster sind im konkreten Einsatz häufig entscheidend für den Grad der Eignung eines Patterns. Diese ‚Kräfte' sind oft gut im Kontext der Problembeschreibung zu erläutern. Auch lässt sich das Problem häufig am besten im Licht einer Reihe von Anforderungen an Software des entsprechenden Anwendungsgebietes beschreiben.

Problem: In diesem Feld wird der erste Teil des eigentlichen Patterns beschrieben, das ‚wiederkehrende Problem'.

Beispiele: Wer nach Patterns sucht, wird häufig aus möglichst unterschiedlichen Anwendungsbeispielen am besten ableiten können, ob ein Pattern geeignet ist.

Lösung: In diesem Feld muss der zweite Teil des eigentlichen Entwurfsmusters beschrieben werden als Lösung zum genannten Problem. '*Real patterns are prose*', wenn also eine formale Beschreibung möglich ist, liegt statt eines Pattern ein Algorithmus (Operation als Teil einer Objektklasse?) oder ein Aspekt vor.

Diagramm: Es ist kein Widerspruch zum eben Genannten, dass eine schematische Darstellung der Lösung fast immer angegeben werden kann. Wer Erfahrung sammelt – hier: wer Patterns lernt oder sucht –, wird sich ein solches Diagramm besser einprägen können als die verbale Lösungsbeschreibung.

Kontext: Wie erwähnt werden Patterns auf verschiedenen Detaillierungsstufen des Entwurfs eingesetzt. Sie kommen bei Systementwurf, Komponentenentwurf und Komponenten-Realisierung (hier als ‚Programming Patterns') vor. Auch Patterns ein und derselben Stufe schließen sich ggf. aus oder beeinflussen sich. Aus all diesen Gründen ist es wichtig, dass der „erfahrene Programmierer" im hier beschriebenen Feld auch seine Erfahrung über die Beziehungen des aktuellen Patterns zu anderen niederschreibt.

Pattern-Sprachen

Innerhalb eines interessierenden Teilbereichs wie der Sicheren Systeme entsteht aufgrund der erwähnten Zusammenhänge zwischen Patterns (sowohl zwischen als auch innerhalb von Detaillierungsstufen) ein konsistentes Gesamtgebilde, quasi ein „Erfahrungsschatz des Entwurfs", wenn die zusammenhängenden Patterns geeignet systematisiert werden. Insbesondere sind dazu konsistente und kohärente Begriffe und Bezeichner zu wählen, weshalb man ein solches Gesamtgebilde auch als Pattern-Sprache (oder Pattern-System) bezeichnet. Selbstverständlich gibt es aber auch Zusammenhänge, die über den Teilbereich hinausgehen, so dass idealerweise sogar ein Geflecht von Pattern-Sprachen entsteht. Abb. 1 illustriert diesen Sachverhalt für interaktives computergestütztes Musik-Schaffen, wie weiter unten noch erläutert wird.

Entwurfsmuster für sichere Systeme

Obwohl Sicherheit heute zu den am häufigsten geforderten Eigenschaft moderner verteilter Anwendungen gehört, sind diese von einem adäquaten Sicherheitsniveau weit entfernt: die gleichen Fehler treten immer wieder auf, wie beispielsweise Buffer Overflows in Anwendungen oder Standardpasswörter in IT Systemen. Während Kodierungsfehler auto-

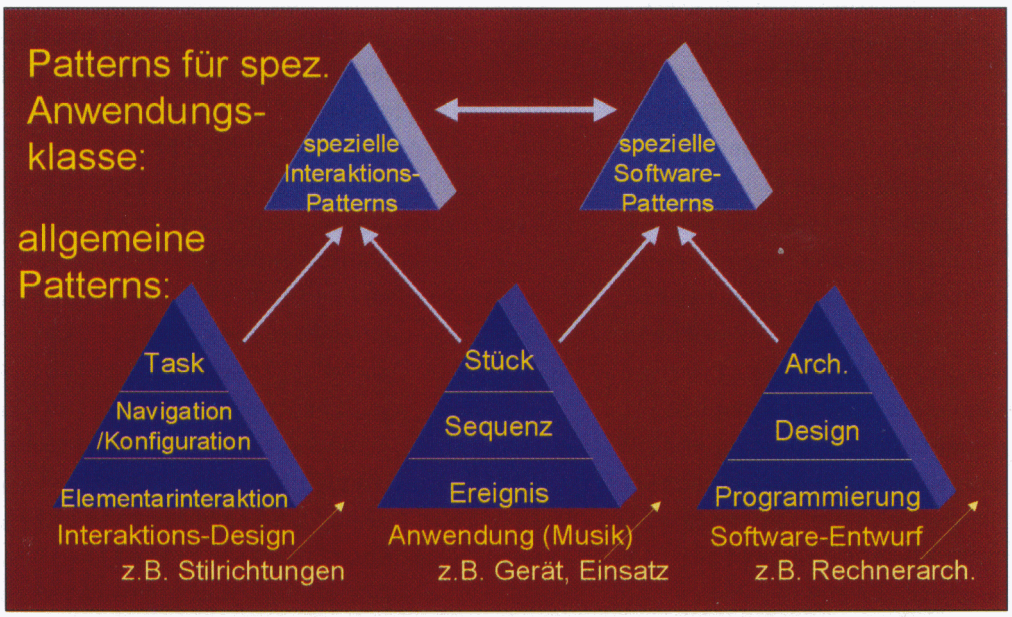


Abb. 1

matisch erkannt werden können, ist es eher schwierig, Sicherheitsaspekte auf der Entwurfsebene sicherzustellen. Gründe dafür sind u.a. im menschlichen Verhalten beim Problemlösen zu suchen: Probleme werden zu stark ad-hoc gelöst, ohne Nebenwirkungen zu berücksichtigen; Termindruck und die ohnehin komplexe Aufgabenstellung verführen Entwickler dazu, Themen wie Sicherheit ‚auszuklammern' etc. Security Patterns wirken dem entgegen, indem sie die Erfahrung von Sicherheitsexperten bei der zuverlässigen Analyse von Software(-Entwürfen) im Hinblick auf Sicherheitslücken systematisch erfassen und damit für unerfahrene Entwickler, aber auch zwischen Sicherheitsexperten kommunizierbar und anwendbar machen. Von der Vielzahl der durch den zweiten Autor dieses Beitrags systematisierten Security-Patterns sei beispielhaft eines in gekürzter Form aufgeführt:

Name: Handhabung von Cookies

Erläuterung: Serveranwendungen verwenden Cookies, um Informationen über Klienten abzuspeichern und abzufragen. Ein Server verwendet den http-Header, um ein Cookie dauerhaft bei dem Klienten zu plazieren und so Informationen wie beispielsweise eine eindeutige Benutzerkennung zu hinterlegen. Fragt der Klient eine weitere Seite von dem Server an, wird das Cookie automatisch mit der http-Anfrage versendet.

Problem: Zum Schutz der Benutzer wurde vereinbart, dass ein Cookie nur von dem Server gelesen werden kann, der es erzeugt hat. Weiterhin dürfen nur Informationen, die entweder vom Klienten oder vom Server erzeugt worden sind, ausgetauscht werden. Diese Konventionen genügen jedoch nicht, um die Privatsphäre des Anwenders zu schützen. Folgende Bedrohungen können identifiziert werden:

- Dienstanbieter können die Aktivitäten eines Benutzers verfolgen und somit Profile erstellen.
- Benutzerdaten verschiedener Anbieter können zusammengeführt werden. Dies geschieht z. B. bei weltweit tätigen Werbungsfirmen wie AdDoubleclick.
- Mit Hilfe von HTML-basierten E-Mail-Nachrichten können Cookies ohne Wissen des Benutzers personalisiert werden.

Kräfte/Anforderungen: Folgende Anforderungen müssen im genannten Kontext berücksichtigt werden:

- Anonymität: Dienstanbietern und anderen Benutzern soll es nicht möglich sein, die Identität eines Benutzers aus einer HTTP-Anfrage abzuleiten. Ein Dienst darf weiterhin nicht den richtigen Benutzernamen offen legen.
- Unverknüpfbarkeit: Ein Benutzer sollte mehrere HTTP-Anfragen stellen können, ohne dass diese miteinander in Verbindung gebracht werden.
- Benutzbarkeit: viele Angebote sind nicht benutzbar, wenn Cookies nicht akzeptiert werden. Außerdem möchten Benutzer oft keine zusätzliche Software kaufen und installieren bzw. Änderungen in der Konfiguration vornehmen.

Lösung: Die Verwendung von Cookies muss auf Seiten des Benutzers eingeschränkt werden. Nahezu alle Web-Browser erlauben es, Cookies von Fall zu Fall zu aktivieren. Der Benutzer muss dann immer entscheiden, ob er einem Anbieter vertraut oder nicht. Cookies können auch periodisch gelöscht werden, so dass die Erstellung von Profilen nicht mehr möglich ist. Diese Variante wird z.B. von Programmen wie Opera oder Junkbuster unterstützt. Maximaler Schutz ergibt sich, wenn Cookies grundsätzlich deaktiviert werden. Der Schutz der Privatsphäre ist immer mit Kompromissen verbunden. Je stärker die Verwendung von Cookies eingeschränkt wird, desto stärker wird die Benutzbarkeit eines auf Cookies basierenden WWW-Angebots beeinträchtigt.

Entwicklung interaktiver Systeme

Anhand des zweiten Anwendungsgebietes soll insbesondere der Einfluss von Anwendungsgebieten auf die Ausprägung von Patterns und deren Zusammenhänge verdeutlicht werden. Der dritte Autor dieses Beitrages hat sich dazu auf interaktive Softwaresysteme, insbesondere auf Software zum (interaktiven) kreativen Musikschaffen, konzentriert. Anwendungsgebiet ist also die Musik, bei deren Schaffen wie bei der Softwareentwicklung streng formale und kreative Elemente zusammenkommen. So erstaunt es wenig, dass auch in der Musik Patterns eine wichtige Rolle spielen. Sie sind in Abb. 1 in die Detaillierungsstufen *Stück, Sequenz und Ereignis* gegliedert, letztere betrachtet z.B. einzelne Noten oder Akkorde. Die Abbildung macht deutlich, dass der Interaktionsentwurf – bisher eher ein ‚Stiefkind' der Softwareentwicklung – eine zentrale eigenständige Rolle bei der Systementwicklung einnehmen sollte. Interaktions- und Software-Entwurf sind als Hierarchien von Patterns dargestellt, bilden also eigene Pattern-Sprachen. Abb. 1 zeigt auch, dass in jeder Pattern-Sprache neben der hierarchischen Verzahnung der Detaillierungsstufen auch andere ‚Dimensionen' in Pattern-Systematiken aufgenommen werden sollten, bei den Musik-Pat-

terns z.B. das Instrument (hier: ‚Gerät', da es um computergestütztes Musikschaffen geht). Die vielleicht wichtigste Erkenntnis der hier diskutierten Forschungsarbeiten zeigt sich in den beiden Pyramiden „im Hintergrund": sowohl Softwareentwurf als auch Interaktions-Entwurf müssen im Zusammenhang mit einem Anwendungsgebiet um anwendungsspezifische Patternsprachen (!) erweitert werden. Dieser Zusammenhang soll zum Abschluss dieses Artikels anhand eines kleinen Beispiels verdeutlicht werden, nämlich eines Software-Patterns, welches sich speziell auf computergestütztes Musikschaffen bezieht, also der Pyramide ‚rechts hinten' zuzuordnen ist. Das zugehörige Diagramm-Feld ist in Abb. 2 wiedergegeben. Da die vollständige Beschreibung den Rahmen dieses Artikels sprengen würde, werden wieder nur einige wichtige Felder verkürzt wiedergegeben.

Name: Metric Transformer (*)

Erläuterung: Computergestütztes Musikschaffen ist oft damit befasst, Benutzerinteraktionen wie Dirigiergesten oder Spielgesten ‚in Realzeit' in Musik umzuformen. Auf der hier besprochenen Ebene werden Teilprobleme dieses Szenarios erfasst.

Problem: Computergenerierte Musikdaten – auch wenn sie von menschlichen Gesten abgeleitet sind – erzeugen bei der unmittelbaren Ausgabe oft einen sterilen Höreindruck. Hierfür ist insbesondere der ‚perfekte' Rhythmus verantwortlich, der beim (Zusammen-)Spiel von menschlichen Musikern nie erreicht wird und z.B. beim ‚Swing' absichtlich variiert wird (Stichwort: ‚Groove').

Lösung: Um einen Datenstrom von Musikdaten ‚menschlich' zu transformieren, kann man ein Subsystem einsetzen, welches aus sechs kooperierenden Objekten besteht, siehe Abb. 2: Der *Modulator* macht dabei dem *Timer* die Vorgaben, wonach die vom *Metronom* erzeugten ‚sterilen' Rhythmen moduliert werden. Falls Benutzer diese Modulation in gewissem Umfang steuern können, ist hierfür eine Interaktionskomponente *Customizer* vorzusehen. Die modulierten Rhythmen gehen an den *Spieler,* welcher aus dem vom Erzeuger gelieferten (ggf. in Realzeit berechneten) Musik-Datenmaterial letztlich die hörbare Musik erzeugt.

An dieser Stelle wird das unvollständige Beispiel-Pattern abgebrochen, da die wesentlichen Gesichtspunkte klar geworden sein sollten. Der Leser mag fragen, wie sich die beiden in diesem Artikel vorgestellten Patterns von Aspekten (insbesomdere das Security-Pattern) bzw. interagierenden Komponenten unterscheiden. Der Unterschied besteht darin, dass hier nicht konkrete Aspekte oder Komponententypen entworfen werden, sondern vielmehr erprobte Prinzipien vorgestellt werden, die bei deren Entwurf und Realisierung angewendet werden können.

Zusammenfassung und Ausblick

Entwurfsmuster als fünfte ‚Stufe' der komponentenorientierten Software-Entwicklung erfassen das, was einen ‚guten' Entwickler wertvoll macht: seinen Erfahrungsschatz. Dieser wird durch die halbformale Darstellung systematischer und leichter erlernbar und über Pattern-Sprachen sowie deren Verflechtung im Zusammenhang verständlich. Die Pattern-Forschung versucht, für immer mehr Teilbereiche und Anwendungsgebiete des Softwareentwurfs Pattern-Sprachen herauszuarbeiten und deren Verflechtung sowie die halbformale Darstellung, Suche und Anwendung von Entwurfsmustern zu verbessern.

Literatur

[BO00] J. Borchers (2001), A Pattern Approach to Interaction Design. Chichester: J. Wiley & Sons.

[SC02] M. Schumacher (2002), Security Patterns, Informatik Spektrum 20, Juni 02. Heidelberg: Springer, pp. 220-223.

[SS00] D. Schmidt, M. Stal, H. Rohnert, F. Buschmann (2000), Pattern-Oriented Software Architecture. Chichester: J. Wiley & Sons.

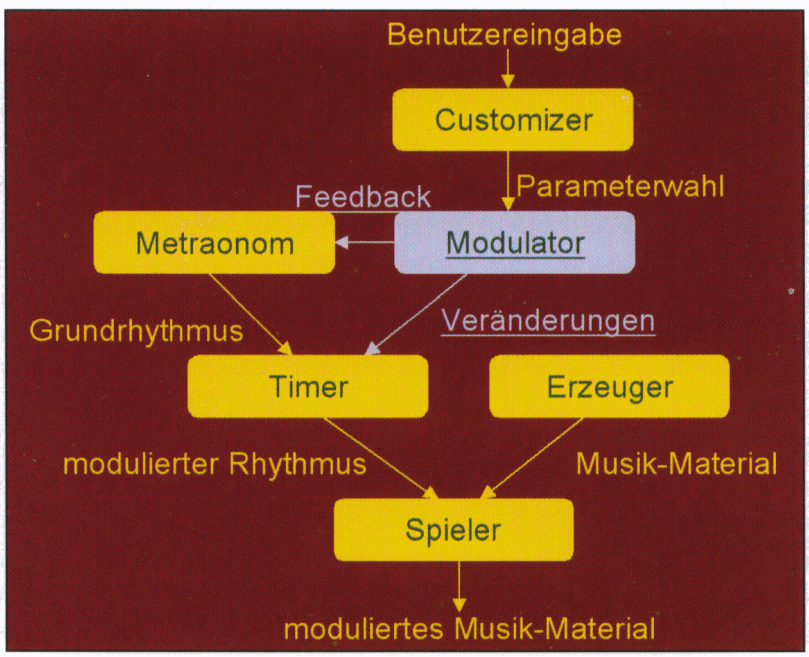


Abb. 2

Vorstellung des Fachgebietes Telekooperation an der TU Darmstadt und des ITO

Das Fachgebiet Telekooperation erforscht, wie Menschen mit und mittels Netzen (‚dem Internet') zusammenarbeiten können. Global oder lokal, drahtlos oder via Breitbandkabel, explosionsartig wachsend, ist das Netz Infrastruktur für die Konvergenz von Informationstechnik, Telekommunikation und „Medien", allgegenwärtig (ubiquitous) und hoffentlich unauffälliger Bestandteil unserer Umwelt (invisible, ambient). Leitlinie der Forschung sind innovative Anwendungen, für deren effiziente Herstellung Werkzeuge, Plattformen und Methoden, programmiersprachliche und Hypertext-Konzepte, sogar Hardware-Prototypen entwickelt werden. Der Forschungsbereich uBiz betrachtet ubiquitous computing für eCommerce, für das Internet als viertes Massenmedium und für vernetzte multimediale Softwaresysteme, insbesondere bei heterogenen Netzen und Betriebsmodellen (UMTS, wLAN-Hotspots etc.). Weitere Arbeiten betreffen innovative Endgeräte und mobile-Commerce, bioanaloge Handhabung von Multimediadaten, Durchstöbern und Editieren riesiger Digitalvideo-Datenbestände, innovatives Musizieren im Internet und mehr. Der zweite Forschungsbereich ubiquitous learning unterstützt mobile Nutzer, die Vollzeit oder lebenslang lernen bzw. mit Wissenserwerb verknüpfte Aufgaben lösen. Klassenzimmer/Hörsäle werden aber nicht nur ‚virtualisiert', sondern – aufgerüstet mit ‚Learning Appliances' und Softwarewerkzeugen – als Katalysator und Ort der Begegnung Wurzel innovativen Lernens.

Das Information Technology Transfer Office (ITO) ist ein am Fachbereich Informatik der TU Darmstadt angesiedeltes Zentrum für angewandte Forschung auf den Gebieten IT Sicherheit, Middleware und Ubiquitous Computing. Seit seiner Gründung wird das ITO zu 100% aus Drittmitteln finanziert. Diese setzen sich aus Kooperationen mit führenden IT- und Software-Unternehmen sowie aus öffentlichen Fördermitteln zusammen (z.B. EU Projekte). Die Stärke des ITO ist der breite Forschungshorizont, in den die Themen und Forschungsgruppen der assoziierten TUD Professoren einfließen.

Zu den Autoren:
Prof. Dr. Mühlhäuser ist Leiter des Fachgebietes Telekooperation und Mitglied im Leitungsgremium des ITO, Markus Schumacher ist Leitender Mitarbeiter des ITO und Dr. Borchers ist inzwischen acting Assistant Professor an der Stanford University.

Ansprechpartner:
Prof. Dr. M. Mühlhäuser,
TU Darmstadt, FB20 Telekooperation,
Alexanderstr. 6, D-64283 Darmstadt;
max@informatik.tu-darmstadt.de