Komfort als Kriterium für die Systementwicklung[1]

Lars Libuda, Nico Hamacher

1 Einleitung

Interaktive Geräte unterliegen heutzutage immer kürzeren Entwicklungszyklen, wobei ihr Funktionsumfang stetig zunimmt. Auf der einen Seite steigen damit die Anforderungen an die Gerätegestaltung, was die Handhabbarkeit der Geräte sowie den Komfort der Benutzung betrifft. Die einfache und intuitive Handhabbarkeit ist heute ein wichtiges Verkaufsargument, da sich der Funktionsumfang eines Gerätetyps von Hersteller zu Hersteller nicht mehr relevant unterscheidet. Auf der anderen Seite steigen auch die Anforderungen an die Entwickler dieser Geräte, die in immer kürzeren Zeitabständen mehr Funktionen und die für die Nutzung der Funktionen notwendige Benutzungsoberfläche implementieren und testen müssen. Zeitdruck führt bei der Entwicklung aber häufig zu Fehlern und verlangsamt die ausreichende Erprobung von Prototypen des Systems.

Heutzutage wird der Entwicklungsprozess durch zahlreiche Werkzeuge unterstützt, die eine automatisierte Systementwicklung und -bewertung erlauben. Dadurch können Entwicklungszeiten beschleunigt und die Wiederverwendbarkeit von Systemkomponenten gewährleistet werden. Solche Werkzeuge entlasten den Systementwickler u.a. durch die Übernahme stetig wiederkehrender Aufgaben und stellen somit einen deutlichen Komfort für den Systementwickler dar (Fels & Hausrotter, 2003).

Trotzdem entwickeln immer noch viele Firmen ihre eigenen Softwarewerkzeuge. Ziel dieses Beitrages ist die detaillierte Betrachtung der Möglichkeiten der werkzeuggestützten Systementwicklung als Komfort für den Systementwickler. Der folgende Abschnitt erklärt den typischen Entwicklungsprozess für interaktive Systeme und erläutert Möglichkeiten der Werkzeugunterstützung während der Systementwicklung. Darauf aufbauend stellen Abschnitt 3 und 4 an Hand zweier Fallbeispiele Möglichkeiten für die Unterstützung der Entwickler bei der Anwendungsentwicklung und Usability-Evaluierung vor. Eine Diskussion in Abschnitt 5 beendet diesen Beitrag.

---

[1] In: 47 Fachausschusssitzung Anthropotechnik „Komfort als Entwicklungskriterium in der Systemgestaltung“ , Volume DGLR-Bericht 2005-05, pp. 19-39, 25-26.Oktober, Wolfsburg, DGLR, ISBN 3-932182-44-8

2 Entwicklung von interaktiven Systemen

Ein interaktives System lässt sich nach dem Seeheim-Modell als logisches Architekturmodell[2], wie in Bild 1 dargestellt, in mehrere Komponenten unterteilen (Pfaff (1983)), wobei diese Struktur heutzutage als Grundlage der meisten dialoggesteuerten Systeme gilt.

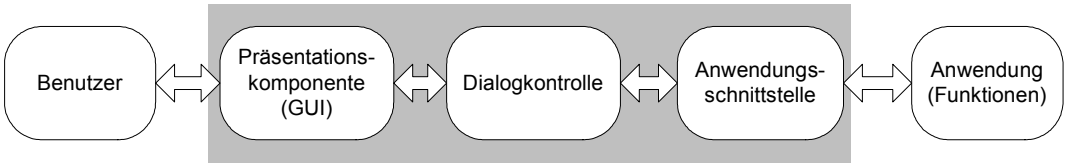

Bild 1: Seeheim-Modell nach Pfaff (1983).

Die *Präsentationskomponente* (bzw. UI oder GUI) dient der Informationsein- und -ausgabe. Ihre Aufgabe besteht in der Bildschirmverwaltung, der Verwaltung der physikalischen Eingabegeräte, der textuellen und grafischen Ausgabe sowie der Realisierung verschiedener Interaktionstechniken. Sie enthält die dem Benutzer sichtbaren und zugänglichen Komponenten einer Benutzungsschnittstelle.

Die *Dialogkontrolle* definiert und überwacht die korrekte Abfolge von Dialogschritten, die durch Interaktionstechniken der Präsentationskomponente realisiert werden und steuert dadurch das dynamische Verhalten einer Benutzungsschnittstelle.

Die *Anwendungsschnittstelle* beschreibt die Systemfunktionalität aus der Sicht der Benutzungsschnittstelle. Sie definiert die für die Benutzungsschnittstelle zugreifbaren Datenstrukturen und Funktionen der Anwendung sowie die Struktur der Informationen, die zwischen der Benutzungsschnittstelle und der Anwendung ausgetauscht werden.

Die *Anwendung* selber entspricht der Implementierung der durch das System angebotenen Funktionalität.

Die einzelnen Phasen der Entwicklung und Bewertung der Komponenten interaktiver Systeme werden mit Hilfe von Phasenmodellen dargestellt. Die Phasen entsprechen jeweils einem Arbeitspaket mit definierten Arbeitsschritten und Teilergebnissen. Das Ergebnis jeder Phase ist die Grundlage der jeweils darauf folgenden Phase. Durch die getrennte Darstellung dieser Phasen wird ein größeres Verständnis ihrer selbst und ihrer Interaktion möglich. Ein phasenorientierter Ansatz stellt eine systematische Darstellung des idealtypischen Ablaufs einer Systementwicklung dar. Er modelliert nicht den realen Prozessablauf. Die Überführung des Modells in ein entsprechend der konkreten

[2] *Logisches Architekturmodell* besagt, dass jede Benutzungsschnittstelle die Funktionalität der drei Komponenten des Seeheim-Modells besitzen sollte, ohne jedoch eine dieser Aufteilung entsprechende monolithische Software-Architektur besitzen zu müssen.

Entwicklungspraxis angepasstes *Working Model* ist jeweils erforderlich. Bild 2 zeigt ein einfaches Phasenmodell nach Kraiss (1995).

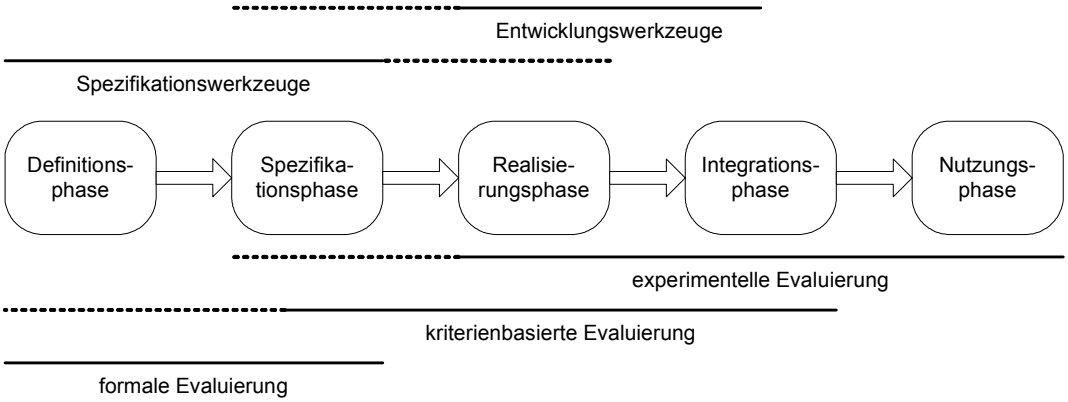

Bild 2: Lebensphasen eines Systems.

Die *Definitionsphase* dient der Definition und Erarbeitung von Leistungsumfang sowie weiteren Anforderungen des zu entwickelnden Systems auf Grundlage einer detaillierten Anforderungsanalyse. Auf Basis der festgelegten Definitionen des Systems wird in der *Spezifikationsphase* die Funktionalität mit allen nötigen (Teil-)Systemen bis ins Detail festgelegt. Das Ziel der *Realisierung* ist die technische Umsetzung des in der Spezifikation festgelegten Systems. Die realisierten (Teil-)Systeme werden in der *Integrationsphase* zum geforderten Gesamtsystem zusammen gefügt. Die *Nutzungsphase* stellt die ordnungsgemäße Anwendung des Systems dar.

Das dargestellte Phasenmodell dient nur der Definition der Phasen. Tatsächlich sind in der realen Systementwicklung zahlreiche Tests und Rückkopplungen von Nöten. Ein Modell zur konkreten Hilfestellung stellt das V-Modell dar, dessen detaillierte Beschreibung mehrere hundert Seiten umfasst (V-Modell, 2005).

3 Unterstützung bei der Anwendungsentwicklung

Bei modernen interaktiven Systemen sind ca. 80 % des implementierten Funktionsumfangs dafür zuständig, die Benutzungsoberfläche intuitiv und komfortabel zu gestalten und fehlerfreie Interaktionen mit den Endanwendern zu garantieren. Nur 20 % der Funktionalität machen die eigentliche Anwendung aus (rechter Block in Bild 1). Allerdings bilden diese 20 % den Kern, der letztendlich die Entwicklung des Gesamtsystems rechtfertigt. Dementsprechend ist auch bei der Anwendungsentwicklung große Sorgfalt auf die Implementierung und Evaluierung zu legen. Dieser Abschnitt zeigt Möglichkeiten auf, wie Entwickler diese Tätigkeit komfortabler ausführen können, um die Entwicklung zu beschleunigen und Fehlerquellen zu verringern.

3.1 Anwendungsentwicklung aus verschiedenen Sichten

Die Anwendungsentwicklung ist nach dem Seeheim-Modell von der Benutzungsoberfläche des Gesamtsystems über die Anwendungsschnittstelle

entkoppelt (s. Bild 1). Daher können sich die Entwickler vollständig auf die Anwendungsentwicklung konzentrieren. Die Anwendung selbst ist sehr häufig ein signalverarbeitendes System, in dem aus den empfangenen Signalen entsprechende Systemreaktionen generiert werden. Dementsprechend ist die Hauptaufgabe der Entwickler die Implementierung und Evaluierung einer Signalverarbeitungskette. Um Anforderungen an ein Entwicklungswerkzeug identifizieren zu können, die den Komfort bei der Anwendungsentwicklung erhöhen, ohne die Entwickler einzuschränken, muss die Entwicklung sowohl aus der Sicht eines signalverarbeitenden Systems als auch eines Entwicklers betrachtet werden.

3.1.1 Systembedingte Kriterien

Signalverarbeitende Systeme sind oft damit konfrontiert, große Datenmengen in möglichst kurzer Zeit verarbeiten zu müssen. Um solche Systeme zu realisieren, wird die benötigte Funktionalität zuerst in hardwarenahen Programmiersprachen umgesetzt. Die Wahl fällt heutzutage meistens auf C bzw. C++. Viele Firmen entwickeln auf dieser Basis eigene Softwarebibliotheken, auf die sie in neuen Projekten zurückgreifen.

3.1.2 Entwicklerbedingte Kriterien

Die Phasen, in denen Entwickler am stärksten involviert sind, sind die Spezifikations-, Realisierungs- und Integrationsphase (Bild 2). Je nach Komplexität der zu realisierenden Anwendung sind mehrere Entwickler an jeder Phase beteiligt, die sich die anfallenden Arbeiten teilen. Dabei treten in den drei Phasen prinzipiell immer die gleichen Aufgaben auf:

In der Spezifikationsphase muss die Anwendung in einzelne Module zerlegt werden. Dabei erfüllt jedes Modul eine Teilaufgabe und kommuniziert über definierte Schnittstellen seine Ergebnisse zu anderen Modulen. Dabei existiert immer ein Modul, das die zu verarbeitenden Signale von Sensoren akquiriert und dem System zur Verfügung stellt. Dieses Modul stellt die Datenquelle dar.

Die Realisierungsphase beinhaltet die Implementierung der einzelnen Module. Diese sind primär durch Algorithmen realisiert, wobei jeder einzelne Algorithmus durch seine Eingangs- und Ausgangsdaten sowie frei wählbare Parameter charakterisierbar ist. Dabei kann es vorkommen, dass für ein Modul mehrere Algorithmen existieren, die die gleiche Teilaufgabe lösen können.

In der Integrationsphase erfolgt die Evaluierung, Optimierung und Zusammenführung der einzelnen Algorithmen. Neu entwickelte Algorithmen enthalten meistens Fehler und sind noch nicht auf Laufzeit und/oder Präzision optimiert. Um Schwachstellen zu entdecken ist es notwendig, die Resultate eines Algorithmus mit repräsentativen Eingangsdaten und verschiedenen Parametereinstellungen zu vergleichen. Ein Vergleich erfordert eine entsprechende Visualisierung der Resultate. Diese Evaluierung ist für jeden Algorithmus durchzuführen und ist dementsprechend zeitaufwändig. Falls die

Evaluierung nicht zufrieden stellend ausfällt, ist eine Rückkehr zur zweiten oder sogar zur ersten Phase notwendig.

3.2 Bestehende Entwicklungswerkzeuge

Die Spezifikationsphase und Realisierungsphase wird durch zahlreiche Werkzeuge abgedeckt (Bild 2). Spezifikationen erfolgen meist auf Basis von UML (Booch et al., 2002), SDL (Ellsberger et al., 1997) oder Statecharts (Harel, 1987). Viele Spezifikationswerkzeuge sind in der Lage aus den Spezifikationen heraus Quellcode zu erzeugen und gestatten somit den direkten Übergang in die Realisierungsphase (z.B. Statemate Magnum (i-Logix, 1997)). Dort stehen zahlreiche integrierte Entwicklungsumgebungen zur Verfügung, die die komfortable Erweiterung und Pflege des erstellten Quellcodes gestatten, z.B. Borlands Delphi (Cantu, 2003).

Der Übergang von der Realisierungs- in die Integrationsphase wird jedoch nur spärlich unterstützt. In diesem Bereich besteht somit Bedarf an Verbesserungen. Der folgende Abschnitt nennt die Anforderungen für ein Werkzeug, das die Realisierungs- und Integrationsphase abdeckt.

3.3 Anforderungen an ein Entwicklungswerkzeug für die Realisierungs- und Integrationsphase

Aus den systembedingten Kriterien und den Aufgaben von Entwicklern in der Integrationsphase lassen sich direkt mehrere Anforderungen ableiten, die ein Werkzeug in dieser Phase erfüllen sollte. Sie sind im Folgenden zusammengefasst.

- **Unterstützung hardwarenaher Sprachen**: Um die Systemkriterien zu erfüllen, muss das Entwicklungswerkzeug die Implementierung von Funktionen in hardwarenahen Sprachen ermöglichen.
- **Unterstützung vorhandener Softwarebibliotheken**: Viele Firmen greifen auf eigene Softwarebibliotheken zurück, mit denen die Entwickler vertraut sind. Diese Bibliotheken müssen in einem Werkzeug verwendbar sein.
- **Zusammenarbeit**: Komplexere Projekte erfordern die Zusammenarbeit mehrerer Entwickler, wobei jeder einzelne Entwickler nur ein Teil des Gesamtsystems realisiert. Trotzdem muss hinterher alles zu einem Gesamtsystem integrierbar sein. Das erfordert eine einheitliche Schnittstelle.
- **Wiederverwendbarkeit von Komponenten**: Einmal programmierte und getestete Komponenten sollten wieder verwendet werden können. Diese Maßnahme steigert die Produktivität und verringert die Fehlerquote. Im Bereich der Signalverarbeitung sollte die Rate von wieder verwendeten Komponenten sehr hoch sein, da hier standardisierte Methoden existieren, die sehr häufig anwendbar sind. Dazu ist es notwendig, dass das Entwicklungswerkzeug die Modularisierung unterstützt.

- **Fokus auf Funktionalität**: Der Fokus der Entwickler sollte auf der Implementierung und Evaluierung der Algorithmen liegen. Die Visualisierung der Resultate und möglichst auch die Akquisition der Eingangssignale aus verschiedenen Quellen sollte das Entwicklungswerkzeug übernehmen, um Entwickler von diesen zeitintensiven Aufgaben zu befreien.
- **Unsichere Spezifikationen**: Zu Beginn der Entwicklungsphase stehen zwar die Spezifikationen des Gesamtsystems fest, können aber geändert werden, falls sich einige Aspekte des Systems unter den gegebenen Spezifikationen als nicht realisierbar herausstellen.
- **Flexibilität**: Das Werkzeug muss flexibel genug sein, um auch in Nachfolgeprojekten eingesetzt werden zu können.
- **Grafische Oberfläche**: Grafische Oberflächen erlauben eine intuitivere Bedienung als Kommandozeilen und bieten große Vorteile bei der Datenvisualisierung.
- **Einfachheit**: Das Werkzeug muss einfach zu nutzen und einfach zu erweitern sein. Voraussetzung dafür ist ein einfaches mentales Modell und eine intuitive Benutzungsschnittstelle.

3.4 Umsetzung der Anforderungen im Entwicklungswerkzeug *Impresario*

Dieser Abschnitt zeigt an Hand des Werkzeugs *Impresario*, das am Lehrstuhl im Bereich der Bildverarbeitung und -analyse eingesetzt wird, wie die im vorangegangenen Abschnitt genannten Anforderungen technisch umsetzbar sind.

3.4.1 Wahl eines Systemmodells

Der erste Schritt bei der Umsetzung ist die Festlegung eines Modells, mit dem ein signalverarbeitendes System beschrieben wird. Das einfachste und sehr weit verbreitete Modell zeigt Bild 4 und besteht aus Filtern, die durch Kanäle miteinander verbunden sind (Buschmann et al., 1996). Beispielsweise arbeitet das DirectShow Interface (Microsoft, 1995) nach diesem Modell.

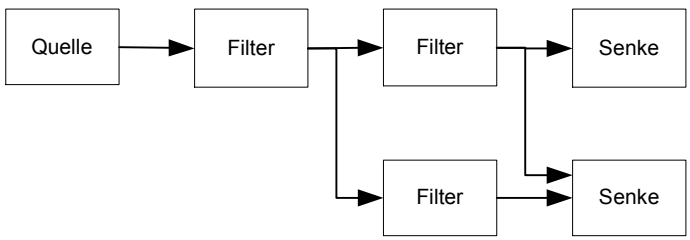

Bild 4: Modellierung einer Signalverarbeitungskette mit Filtern und Kanälen. Die Pfeile kennzeichnen die Kanäle und bestimmen den Datenfluss.

Filter sind die kleinsten Einheiten, die Daten an ihren Eingängen empfangen, diese verarbeiten und an ihren Ausgängen zur Verfügung stellen. Die Kanäle verbinden die Ausgänge eines Filters mit den Eingängen eines nachfolgenden Filters und legen somit den Datenfluss fest. Dieser erfolgt immer von der Quelle zur Senke. In einem Modell sind beliebig viele Quellen, Filter und Senken erlaubt.

Bild 5 zeigt die Schnittstellen eines Filters. Jedes Filter verfügt über beliebig viele Eingänge, Ausgänge und Parameter.

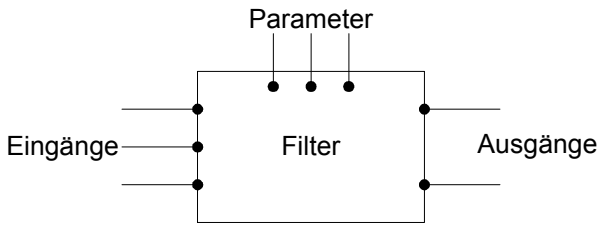

Bild 5: Schnittstellen zu einem Filter. Eingänge, Ausgänge und Parameter sind von außen sichtbar und zugänglich.

Ein- und Ausgänge beinhalten dynamische Daten, die sich während der Laufzeit ändern. Um mehrere Filter korrekt miteinander verschalten zu können, sind Metainformationen über ein- und ausgehende Daten notwendig. Dazu muss jeder Ein- und Ausgang den zu verarbeitenden bzw. erzeugten Datentyp und einen Bezeichner bereitstellen. Im Gegensatz dazu sind Parameter statische Daten, die die Konfiguration eines Filters bestimmen. Sie müssen während der Laufzeit interaktiv geändert werden können. Um das zu ermöglichen, müssen auch sie Metainformationen an das Werkzeug weitergeben. Dazu gehören der Datentyp und Bezeichner sowie ein Standardwert und optional ein Wertebereich.

Das vorgestellte Modell erfüllt mehrere Anforderungen: Es ist sehr einfach aber gleichwohl für die Entwicklung signalverarbeitender Systeme geeignet. Außerdem impliziert es durch die Filter einen modularen Aufbau und eine standardisierte Schnittstelle. Das sind die notwendigen Voraussetzungen zur Erfüllung der Wiederverwendbarkeit von Komponenten sowie die Zusammenarbeit mehrerer Entwickler. Mit Hilfe des Modells und den Metainformationen über Eingänge, Ausgänge und Parameter von Filtern ist es zudem möglich, eine Benutzungsoberfläche zu realisieren, mit dem ein signalverarbeitendes System grafisch erstellt, ausgeführt und getestet werden kann, ohne dass sich Entwickler um Aspekte des zu Grunde liegenden Betriebssystems, Speicherallokation, verwendete Widgetbibliotheken oder Multithreading kümmern müssen. Sie können sich auf die Implementierung der Filter konzentrieren.

3.4.2 Umsetzung des Systemmodells in *Impresario*

Die Umsetzung des gewählten Systemmodells erfolgt in der Sprache C++. Somit ist die Verwendung hardwarenaher Sprachen bei der Entwicklung von Filtern möglich und eine einfache Eingliederung in das Werkzeug gegeben. Zudem ist durch die Verwendung von C++ eine höhere Verarbeitungsgeschwindigkeit möglich als bei der Verwendung interpretierender Sprachen (z.B. Java). Bild 6 zeigt die Benutzungsoberfläche von *Impresario*, an Hand derer die weiteren Konzepte erläutert werden.

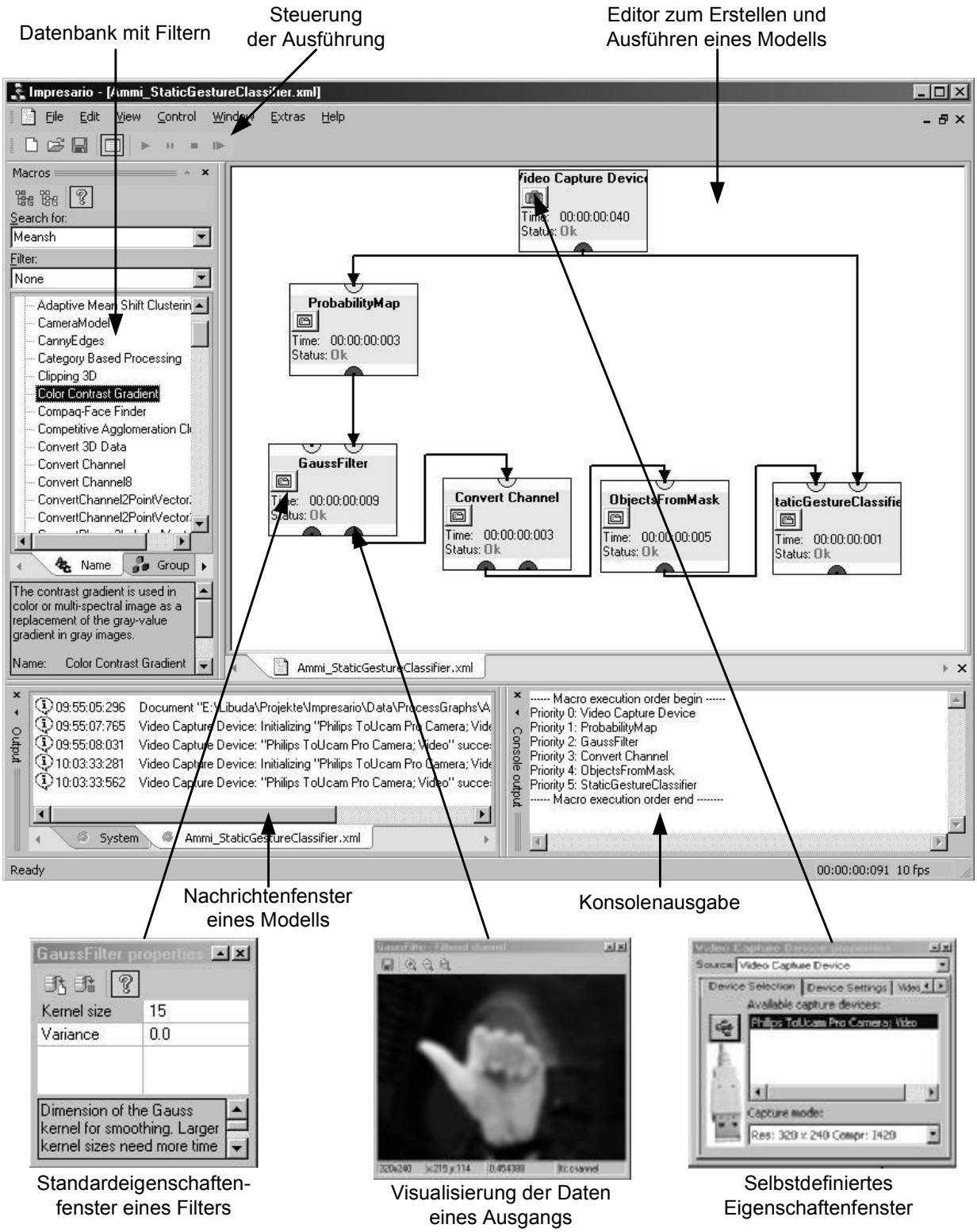

Bild 6: Benutzungsoberfläche von *Impresario* mit einem Beispielmodell zur Unterscheidung statischer Einhandgesten.

*Impresario*s Kernkomponenten sind die Filterdatenbank und der Modelleditor. Damit lassen sich signalverarbeitende Systeme nach dem vorgestellten Modell mittels Drag & Drop Operationen erstellen und ausführen. Modelle können zudem im XML-Format im- und exportiert werden. Ein Filter ist im Editor durch

einen rechteckigen Block repräsentiert. Gemäß dem Filtermodell aus Bild 5 besitzt es Eingänge (oben) und Ausgänge (unten). Zusätzlich besitzt jedes Filter eine Schaltfläche, über die ein separates Fenster zugänglich ist, das die Filterkonfiguration anzeigt. Standardmäßig sind das die Parameter eines Filters (Bild 6, unten links). Allerdings können Entwickler bei Bedarf diese Anzeige durch eigene ersetzen (Bild 6, unten rechts). Jeder Filterblock zeigt zudem Informationen über seinen aktuellen Zustand und die letzte Ausführungsdauer an. Mit Hilfe von Betrachtern lassen sich Daten an den Filterausgängen visualisieren (Bild 6, unten mittig). Betrachter visualisieren Daten eines bestimmten Datentyps. Da die Filterausgänge beliebige Datentypen enthalten können, ist es notwendig, Betrachter nachträglich in *Impresario* einbauen zu können.

Aus diesen Gründen existieren zwei Programmierschnittstellen, mit denen *Impresario* erweiterbar ist:

- **Schnittstelle für Filter**: Das ist die Standardschnittstelle, um neue Filter zu *Impresario*s Datenbank hinzuzufügen. Sie wird in Abschnitt 3.4.3 genauer betrachtet.
- **Schnittstelle für Betrachter**: Mit dieser Schnittstelle lassen sich neue Betrachter für bislang unbekannte Datentypen in *Impresario* integrieren.

Filter und Betrachter werden in von der Hauptanwendung getrennten, dynamisch ladbaren Programmdateien[3] gespeichert und beim Start von *Impresario* eingelesen.

Mit Hilfe dieses Plugin-Konzepts lassen sich mehrere Anforderungen erfüllen. *Impresario* ist durch die beiden Schnittstellen völlig unabhängig von jeglichen Spezifikationen, die das zu realisierende signalverarbeitende System zu erfüllen hat, solange das System auf das zu Grunde liegende Modell abzubilden ist. Es ist daher sehr flexibel und z.B. auch im Bereich der Sprachverarbeitung einsetzbar. In Kombination mit *Impresario*s Datenbank wird auch die Zusammenarbeit mehrerer Entwickler unterstützt. Jeder Entwickler kann eigene Programmdateien mit Filtern erzeugen, die nachher in *Impresario* zusammengelegt werden, um alle Filter für das Gesamtsystem verfügbar zu haben. Zusätzlich garantiert die Datenbank die Wiederverwendbarkeit schon implementierter Filter und Betrachter in späteren Projekten. Letztendlich sorgen die zwei Programmierschnittstellen für eine strikte Trennung von Funktionalität und Visualisierung und somit für die Möglichkeit, die Implementierung der Funktionalität in den Vordergrund zu rücken.

Der folgende Abschnitt zeigt, wie einfach die Einbindung neuer Filter in *Impresario* ist.

---

[3] Unter dem Betriebssystem Windows werden die dynamisch ladbaren Programmdateien als Dynamic Link Libraries (DLL) bezeichnet.

3.4.3 Programmierschnittstelle für Filter

Jedes Filter muss mindestens die in 3.4.1 genannten Metainformationen enthalten, um die Integration in *Impresario* zu gewährleisten. Somit ist eine API[4] notwenig, an die sich die Entwickler halten müssen. Allerdings sollte die API den Mehraufwand bei der Entwicklung eines Filters so gering wie möglich halten, damit auch hier die Funktionalität im Vordergrund steht.

Um dieses Ziel zu erreichen, muss soviel wie möglich automatisiert werden. Das geschieht in *Impresario* unter Zuhilfenahme objektorientierter Konzepte. Jedes Filter stellt eine Klasse im Sinne der objektorientierten Programmierung dar. Als API dient eine Basisklasse, die Methoden zur Beschreibung der Ein- und Ausgänge sowie der Parameter bereitstellt. Weitere Methoden dienen der Verarbeitung sowie der Erzeugung eigener Konfigurationsfenster (Bild 6, unten rechts). Neue Filter leiten sich von dieser Basisklasse ab und rufen deren Methoden auf oder überschreiben sie.

Somit steht eine sehr einfache Schnittstelle zur Verfügung, die jedoch genug Flexibilität bietet, um firmeneigene Softwarebibliotheken zu unterstützen und Eingriffe in *Impresario*s Benutzungsschnittstelle zu realisieren. Letzteres ist immer optional.

Der Codeausschnitt in Bild 7 zeigt den minimal notwendigen Aufwand, um ein neues Filter zu erzeugen. Das Beispiel zeigt die Realisierung des in Bild 6 enthaltenen Gaussfilters und verwendet dafür die lehrstuhleigene Bildverarbeitungsbibliothek LTI-Lib[5]. Mit Hilfe von Skripten kann der Großteil des Codes automatisch generiert werden.

3.5 Praktischer Einsatz von *Impresario* im Bereich der Bildverarbeitung

Mit *Impresario* wurden bereits mehrere kleine und große Anwendungen aus dem Bereich der Bildverarbeitung realisiert. Das Modell in Bild 6 zeigt beispielsweise ein System zur Unterscheidung von drei statischen Einhandgesten. Es dient als Klassifikationsbeispiel für (Kraiss, 2006), in dem *Impresario* als Demonstrationssoftware enthalten sein wird. Weiterhin dient es als Forschungsplattform im Rahmen der automatischen Szenenanalyse in Innenräumen (Libuda, Kraiss, 2004).

Alle Filter eines Modells werden gemäß dem Datenfluss sequentiell ausgeführt. Auch Rückkopplungen zwischen Filtern sind dabei berücksichtigt. Allerdings müssen die Filter selbst dafür sorgen, dass ihre initialen Ausgangswerte im erwarteten Gültigkeitsbereich liegen.

---

[4] API: Application Programming Interface

[5] Die LTI-Lib ist ein Open-Source-Projekt des Lehrstuhls und unter http://ltilib.sourceforge.net frei erhältlich.

```
#ifndef GaussFilter_h
#define GaussFilter_h

#include "macrotemplate.h"   // Klasse mit Interface
#include "ltiConvolution.h"
#include "ltiImage.h"

// Definition des Gaussfilters als Klasse, die von
// CMacroTemplate abgeleitet ist.
class MACRO_API CGaussFilter : public CMacroTemplate {
public:
  // Standardkonstruktor und -destruktor
  CGaussFilter(void);
  virtual ~CGaussFilter(void);

  // Methode zum Ausführen des Filters ("Hauptprogramm")
  virtual bool Apply();
  // Methode zum Anlegen eines neuen Filters diesen Typs
  virtual CMacroTemplate* Clone() {
    return new CGaussFilter();
  }

private:
  // Klassenattribut, das als Eingang für das Filter dient.
  const lti::channel*  m_pchnInput;
  // Klassenattribut, das als Ausgang für das Filter dient.
  lti::channel         m_chnOutput;
  // Faltungsalgorithmus aus der LTI-Lib
  lti::convolution     m_fctGaussFilter;
};

#endif
```

```
#include "GaussFilter.h"
#include "ltiGaussKernels.h"

CGaussFilter::CGaussFilter(void) {
  // Deklaration des Eingangs
  AddMacroInput("Channel","lti::channel for gauss filtering.",&m_pchnInput);
  // Deklaration des Ausgangs
  AddMacroOutput("Filtered channel","Filtered lti::channel.",&m_chnOutput);
  // Deklaration von 2 Parametern
  AddMacroParameter(tInt,"Kernel size",
    "Dimension of the Gauss kernel for smoothing. Larger kernel sizes "
    "need more time to be calculated but smooth the image stonger than "
    "smaller kernels."
    ,static_cast<int>(5),static_cast<int>(1),static_cast<int>(10000));
  AddMacroParameter(tFloat,"Variance",
    "Variance of the Gauss kernel. A value of 0.0 calculates a variance "
    "dependent on the kernel size",0.0f,0.0f,1000.0f);
  // Initialisierung des Faltungsalgorithmus aus der LTI-Lib
  lti::convolution::parameters gaussFilterParameters;
  gaussFilterParameters.boundaryType = lti::modifier::parameters::Constant;
  m_fctGaussFilter.setParameters(gaussFilterParameters);
}

CGaussFilter::~CGaussFilter(void) {
}

bool CGaussFilter::Apply() {
  // Auswertung der Parameter und Initialisierung des
  // Faltungskernels.
  lti::gaussKernel2D<float> kernel(GetParameterValue<int>(0),
                                   GetParameterValue<float>(1));
  m_fctGaussFilter.setKernel(kernel);
  // Ausführen der Faltung. Gibt im Fehlerfall false zurück.
  return m_fctGaussFilter.apply(*m_pchnInput,m_chnOutput);
}
```

Bild 7: Minimaler Aufwand zur Realisierung eines Filters in *Impresario* unter Einbindung einer Fremdbibliothek.

4 Unterstützung bei der Usability-Bewertung von Benutzungsoberflächen

Eine Bewertung der Gebrauchstauglichkeit umfasst die Präsentationskomponente und die Dialogkontrolle (s. Bild 1), da Sie direkten Einfluss auf das Erscheinungsbild des Gerätes für den Benutzer haben.

4.1 Klassifizierung von Usability-Methoden

Zur Bewertung der Gebrauchstauglichkeit (Usability) existieren zahlreiche Methoden, die sich in drei Kategorien klassifizieren lassen, in unterschiedlichen Phasen der Systementwicklung Anwendung finden und in Bild 2 dargestellt sind:

Formale (analytische) Methoden basieren meist auf normativen Modellen des Benutzers bzw. dessen Verhalten und lassen sich daher sehr früh im Entwicklungsprozess einbinden. Eine Analyse dieser Modelle kann z.B. Abschätzungen der Ausführungs- und Lernzeit bei der Bearbeitung einer bestimmten Aufgabe liefern und lässt sich automatisch durchführen (Hamacher et al. 2002).

Kriterienorientierte Methoden fassen Erfahrungswissen von Experten in Form von Kriterienkatalogen[6] zusammen. Kriterienkataloge differieren im Fokus der Bewertung, im Umfang bzw. in der Anwendung sehr stark. Sie reichen von der allgemeinen Definition der Gebrauchstauglichkeit (z.B. ISO-Norm 9241) bis hin zu detailliert festgelegten Gestaltungsregeln einer Bildschirmoberfläche (z.B. die Gestaltungsrichtlinien des OS-X von Apple (2005)). Sie existieren daher in nahezu allen Phasen der Systementwicklung.

Mit *experimentellen* Methoden wird ein Prototyp des Systems von Probanden bedient und anschließend bewertet. Eine Bewertung stützt sich auf objektiv erhobene Daten, z.B. Protokolle der Handlungen der Probanden oder Videoaufzeichnungen, bzw. subjektiver Aussagen der Probanden, die mit Hilfe von Interviews oder Fragebögen erfasst werden. Diese Bewertungsmethoden sind sehr zeit- und kostenintensiv und eignen sich daher nur bedingt zur häufigen Durchführung. Zudem benötigen Sie einen lauffähigen Prototyp, der erst in späteren Phasen der Systementwicklung zur Verfügung steht.

4.2 Eigenschaften der Guideline-Bewertung

Das meiste Wissen über Gebrauchstauglichkeit liegt heutzutage bereits in Form von Guidelines vor. Zudem sind die am häufigsten durchgeführten Bewertungen der Gebrauchstauglichkeit Bewertungen aufgrund von Guidelines (Timpe et al., 2002).

Durch die stetige Zunahme der Systemfunktionalität und der Vereinheitlichung der Bedienung und des User-Interfaces (UI) nimmt der Grad der Standardisierung

---

[6] Die Begriffe *Kriterienkatalog* und *Guideline* werden in dieser Arbeit synonym benutzt.

ebenfalls zu. Dadurch werden die Kriterienkataloge immer umfangreicher, weshalb gleichzeitig auch die Anforderungen an den Entwicklungsingenieur steigen, da alle diese Kriterienkataloge bei der Systementwicklung Berücksichtigung finden müssen. So zeigte eine groß angelegte Untersuchung, dass sich 96,1 % der deutschsprachigen Webseiten nicht an die essenziellen W3C-Standards halten (Bleich, 2005).

Zudem sind Guidelines wie z.B. die ISO-Norm 9241 meist nur allgemein gehalten und somit nicht direkt auf ein konkretes Problem hin anwendbar. Daher existiert ein mehr oder weniger großer Interpretationsspielraum. Aufgrund dieser Tatsachen werden Kriterien bzw. Kriterienkataloge nicht immer richtig angewendet oder gar nicht erst berücksichtigt. So machen Experten Fehler oder hatten Probleme mit 91 % vorgegebener Guidelines (Souza & Bevan, 1990).

Die Entwicklung interaktiver Systeme wird zunehmend durch Softwaretools unterstützt (s. Kapitel 3), daher stehen viele Systeminformationen bereits frühzeitig im Entwicklungsprozess elektronisch zur Verfügung. Eine automatische Überprüfung der elektronisch vorliegenden Systeminformationen auf Guideline-Konformität bringt somit nicht nur den Vorteil der komfortablen Verwaltung von Kriterienkatalogen sondern gewährleistet die objektive Anwendung der Kriterien und steigert dadurch die Nachvollziehbarkeit von Bewertungsergebnissen. In den letzten Jahren entstanden daher vermehrt Werkzeuge zur automatischen Bewertung von Kriterien.

Häufig werden Systemdaten mit Vorgabewerten verglichen. Für eine hohe Akzeptanz des Systems bei den Endbenutzern ist eine klare und eindeutige textuelle Informationen z.B. für Überschriften und Menüeinträge sehr wichtig. So widmet sich der Teil 12 der ISO 9241 ausschließlich der Darstellung von Informationen. Um Begriffe sowie ihre semantischen Beziehungen zueinander bewerten zu können, muss eine entsprechende Referenz als Bewertungsbasis zur Verfügung stehen. Ontologien[7] beschreiben geeignet solche Begriffe und ihre Relationen zueinander. Daher eigenen sich Ontologien als Referenz für eine automatische Bewertung.

Um eine automatische Überprüfung der Regeln realisieren zu können, müssen diese operationalisierbar sein. Das bedeutet, dass sie interpretationsfrei und algorithmisch implementierbar sind, mit eindeutigem Bezug auf konkrete Elemente des zu bewertenden Systems. Es existieren einige Empfehlungen zur Prozedur der Operationalisierung, z.B. Beirekdar et al. (2002).

[7] Ontologien beschreiben Dinge und deren Beziehung zueinander. Im hier vorliegenden Kontext der Begriffs-Überprüfung muss die Referenz-Ontologie Begriffe und ihre semantische Zugehörigkeit in Form einer Begriffshierarchie abbilden. Dabei muss ein Begriff semantisch die unter ihm angeordneten Begriffe beschreiben. Ein Beispiel dafür bilden Menüs von Softwareprogrammen oder interaktiven Geräten.

4.3 Werkzeuge zur Guideline-Bewertung

Es existieren mehrere Möglichkeiten der Automatisierung der Usability-Bewertung mit Guidelines. Die einfachste Hilfestellung bieten Werkzeuge zur Verwaltung der Guidelines, wie z.B. MetroWEB (Mariage et al., 2005). Eine größere Unterstützung bieten Werkzeuge zur Prüfung von Systemdaten (z.B. GUI-Informationen) auf Guidelinekonformität, wie z.B. das Werkzeug „Kwaresmi“ für Webseiten (Beirekdar et al., 2005).

Diese Werkzeuge ermöglichen jedoch lediglich eine statische Analyse durch wenige, hart in den Programmquelltext einkodierte Regeln. Der grundlegende Aufbau dieser Werkzeuge ist in Bild 8 dargestellt. Nach der Extraktion der GUI-Daten aus der Anwendung werden diese in unterschiedlichen Prozeduren analysiert. Dabei korrespondiert eine Prozedur i.d.R. jeweils mit einer Guideline. Das Ergebnis dieser Bewertung steht anschließend als Report zur Verfügung.

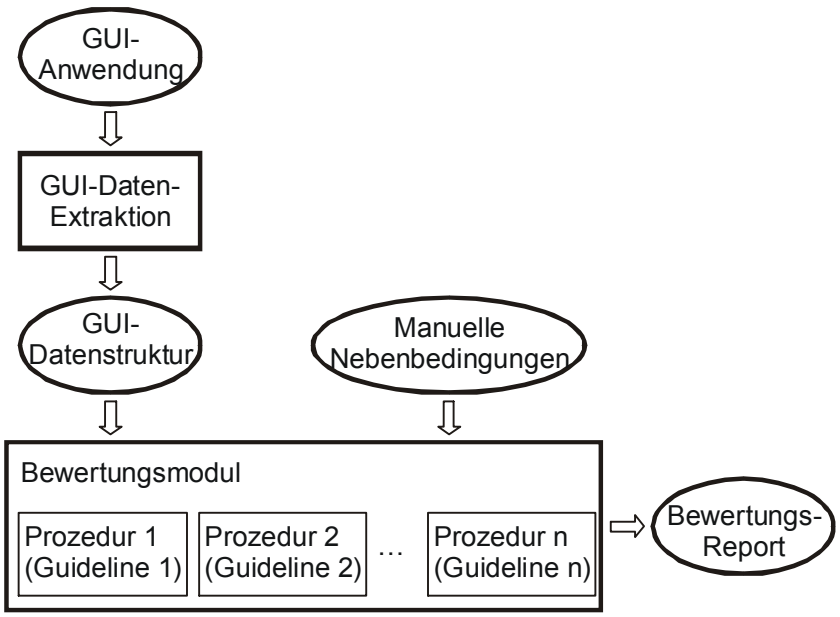

Bild 8: Softwarearchitektur herkömmlicher automatischer Bewertungs-Werkzeuge auf Basis von Guidelines

Der Nachteil dieser Vorgehensweise liegt im statischen Aufbau sowie den eingeschränkten Bearbeitungsmöglichkeiten der Regeln. Zwar lassen sich manuell Nebenbedingungen für die einzelnen Prozeduren festlegen, eine Anpassung bestehender oder Erstellung neuer Regeln ist jedoch nur durch eine aufwändige Änderung des Programmcodes möglich. Zudem liegt die Anwendung dieser Werkzeuge bislang lediglich in der Gestaltüberprüfung der GUI.

4.4 Guideline-Bewertung mit Hilfe von *Reviser*

In diesem Kapitel wird das Werkzeug *Reviser* (Rapid Evaluation of Interactive Systems using Expert Knowledge) vorgestellt, das eine Bewertung mit Hilfe eines Expertensystems realisiert und dessen Architektur in Bild 9 abgebildet ist.

4.4.1 Einbinden von Daten des zu bewertenden Systems in *Reviser*

Eine Beschreibung des GUI des zu überprüfenden Dialogssystems, z.B. aus technischen Spezifikationen oder einem Prototyp, die in der Spezifikationsphase definiert wurden, liegt als Bewertungsbasis zu Grunde. *Reviser* arbeitet intern mit einem eigenen Datenformat zur Beschreibung des System-GUI. Eine Möglichkeit der Einbindung beliebiger Konvertierungsmethoden ermöglicht das flexible Einlesen beliebiger Beschreibungsformate.

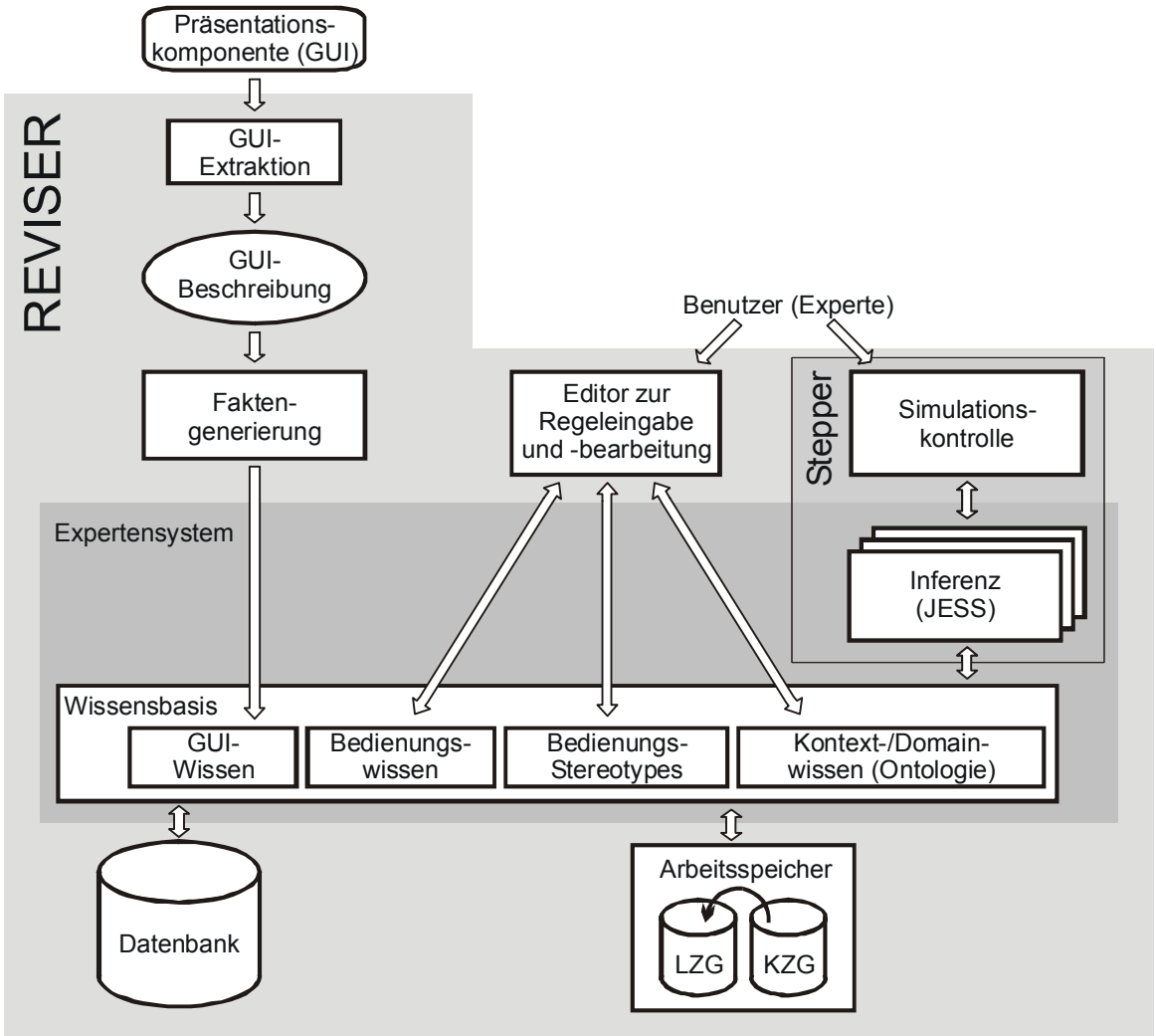

Bild 9: Softwarearchitektur von *Reviser*

In *Reviser* ist ein Expertensystem integriert, dem nach der Konvertierung die Dialogbeschreibung als Fakten zur Verfügung stehen. Dabei kommt das Expertensystem *JESS*[8] (Java Expert System Shell – das Java-Derivat des bekannten CLIPS) zum Einsatz.

[8] http://herzberg.ca.sandia.gov/jess/

4.4.2 Eingabemöglichkeit der Regeln in *Reviser*

Ein Editor ermöglicht die komfortable Eingabe der Regeln der entsprechenden Guideline, dargestellt in Bild 10. *Reviser* erlaubt die hierarchische Anordnung von Regeln in einem Regelbaum. Jeder Knoten des Baumes wird *Step* genannt, wobei jeder Step idealerweise genau einer Regel entspricht.

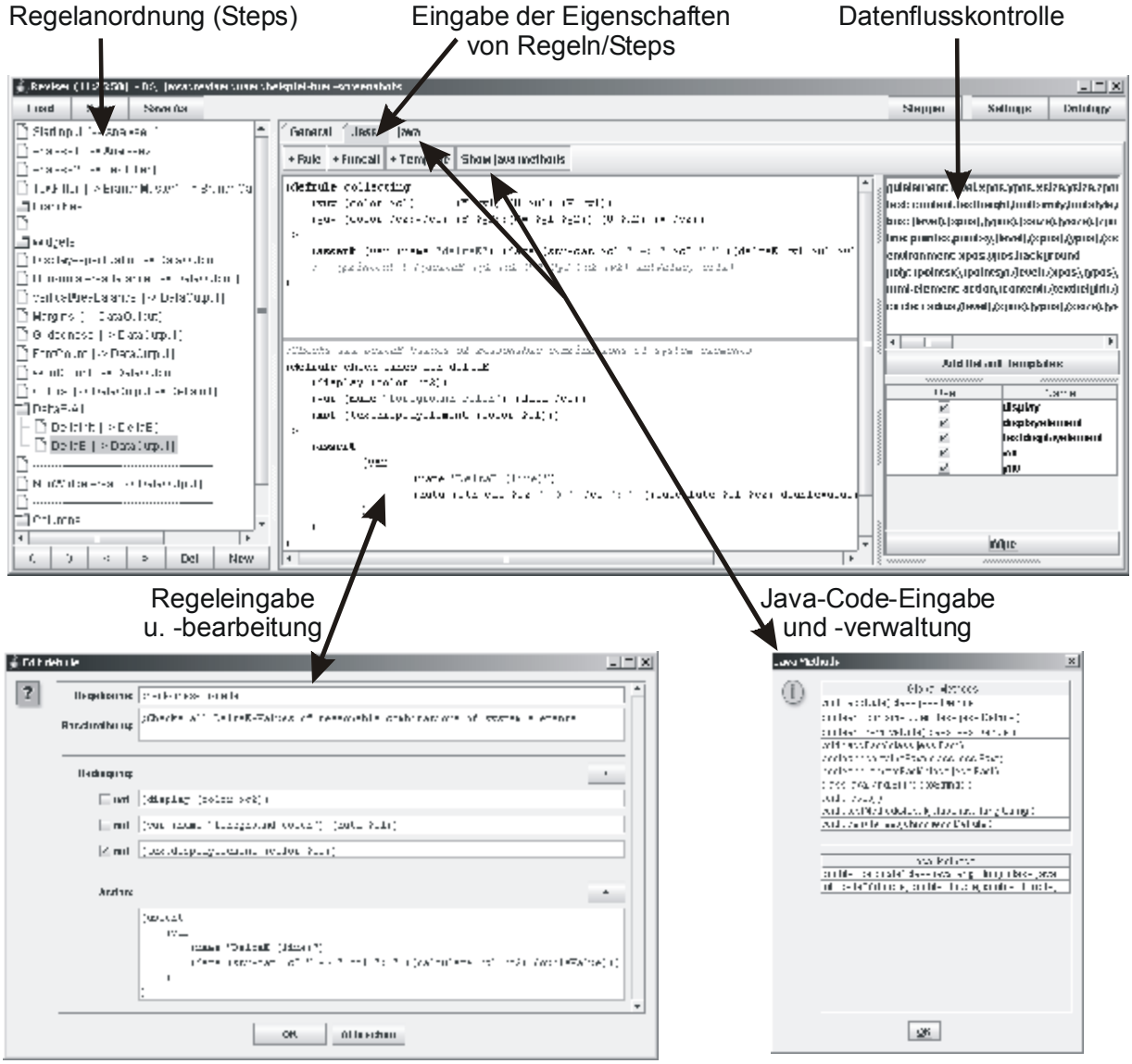

Bild 10: Benutzungsoberfläche von *Reviser* mit Regeleingabedialog und Anzeigedialog der eingebundenen Java-Methoden.

Jeder Step erlaubt eine Kontrolle der Daten, die er von vorangehenden Steps bekommt (rechter Teil in Bild 10). Daten werden in *Reviser* grundsätzlich in Form von Fakten repräsentiert. Fakten haben einen vorgegebenen Aufbau, somit ist es jedem Step auch möglich, neue Fakten zu generieren bzw. eine neue Faktenstruktur (Templates) zu definieren (rechts oben in Bild 10).

Regeln werden in Form von WENN-DANN-Konstrukten (bzw. Bedingung-Aktion) implementiert, wobei Bedingungen grundsätzlich der Prüfung auf

Existenz von Fakten entsprechen und logisch miteinander verknüpft werden können. Die Syntax entspricht der der logischen Programmiersprache LISP.

Zur Integration komplizierter Bewertungsalgorithmen kann Javacode eingebunden und kompiliert werden. Die implementierten Methoden sind in den Regeln ohne jede Einschränkung verfügbar. Eine Übersicht der aktuellen Java-Methoden lässt sich ebenfalls aufrufen (Bild 10, unten rechts).

4.4.3 Ausführung der Bewertung in *Reviser*

Um die Ausführung der einzelnen Regeln zu steuern, ist es möglich, in *Reviser* die Ausführungsreihenfolge anzugeben. Dazu können Steps beliebig miteinander kombiniert werden, Rückkopplungen sind ebenfalls erlaubt.

Im Modul *Stepper* werden die Steps der Reihenfolge gemäß ausgeführt, Daten importiert, Regeln ausgewertet, Javamethoden ausgeführt und entsprechende Ausgaben angezeigt. Die jeweils aktuelle Fakten- und Regelbasis eines jeden Steps lässt sich ebenfalls darstellen.

4.4.4 Simulation der Bedienung in *Reviser*

Bei der Bearbeitung einer Aufgabe werden von einem interaktiven System i.d.R. nach jedem Bedienschritt die Informationen auf dem GUI angepasst (im folgenden *Szene* genannt).

Um die Szenen einer Aufgabe automatisch zu erhalten, muss eine Möglichkeit gegeben sein, das System automatisch zu bedienen. Darüber hinaus ist neben der Bewertung der Gestaltung ebenso die Bewertung der Bedienung von Interesse. Eingabeelemente (Tasten, Stellteile) sollen so stets kontaktanalog bzw. konsistent eingesetzt werden (Hoffmann, 1997).

Reviser unterstützt durch seinen Aufbau die Implementierung eines Bedienagenten. Dieser basiert auf gegebenen und allgemein gültigen Bedienstereotypen und -metaphern sowie einer allgemein gültigen Begriffsontologie zur Orientierung. Dadurch ist er in der Lage, Dialogsysteme automatisch zu bedienen und die Daten der einzelnen Szenen zu speichern (Hamacher & Kraiss, 2004), wofür die Speicher von *Reviser* zum Einsatz kommen (s. Bild 9).

4.5 Beispiel für die Guideline-Bewertung von Fahrerinformationssysteme

Zur Demonstration der Leistungsfähigkeit von *Reviser* sollen zwei Fahrerinformationssysteme (FIS), AUDI MMI und BMW i-Drive, automatisch auf Guideline-Konformität überprüft werden. Die beiden Systeme sind dazu beispielhaft in einem Spezifikationswerkzeug nachmodelliert worden, so dass die Systemdaten elektronisch zur Verfügung stehen.

Bei den zu überprüfenden Regeln handelt es sich um eine Auswahl sowohl der von Sears benutzten Regeln (Sears, 1995) als auch der Human Factor Guidelines des US Department of Transportation (1998). Dabei werden sowohl Regeln

implementiert, die Alarme auslösen (bei Über-/Unterschreitung vorgegebener Werte) als auch Regeln, die einfache Kennzahlen berechnen. Eine Auswahl der implementierten Maße ist im Folgenden gegeben:

- Aspect-Ratio: Verhältnis zwischen Breite und Höhe des Displays
- Gridedness: Maß der „Angeordnetheit“ (Anzahl der GUI-Elemente im Verhältnis der Anzahl der unterschiedlichen Kanten aller Elemente)
- Farben: Anzahl Vordergrund-/Hintergrundfarben
- Delta-E (ΔE): Kontrastverhältnis zweier Farben, ein gutes Kontrastverhältnis besteht für ΔE > 100.
- Non Widget-Area: Unbenutzte Fläche des Displays
- Vertikale und Horizontale Balance: Oben-Unten- sowie Rechts-Links-Ausgewogenheit der GUI-Elememte
- Widget Density: Objekt-Dichte der GUI-Elemente

Es wurden die Szenen von drei Aufgaben bewertet. Bei den Aufgaben handelte es sich um eine Senderwahl beim Radio, eine Klangeinstellung und der Auswahl eines Navigationsziels. Nach jedem Bedienschritt wurden mit Hilfe der implementierten Regeln in ***Reviser*** die genannten Maße für alle angezeigten Informationen im Display berechnet.

Die Ergebnisse der Bewertung sind in Tabelle 1 und im Folgenden gegeben:

	Audi	BMW
Benutzte Farben (Gesamt)	10	8
ΔE-Alarme (ΔE<100)	3	3
Schritte gesamt (Klang, Sender, Navi)	18 (7,5,6)	44 (13,9,22)
Menü-Überschriften automatisch erkannt	ja	nein

Die berechneten Maße wurden anschließend einer Konsistenzanalyse unterzogen. Dabei handelt es sich um die Berechnung des jeweiligen V-Werts, als Maß der mittleren Variation (V):

$$V(Maß) = \frac{\sigma}{E(Maß)} \bullet 100 \qquad (1)$$

σ Standardabweichung des entsprechenden Maßes

Je kleiner der V-Wert, desto enger liegen die Werte des Maßes zusammen und desto konsistenter sind die Werte (Ivory, 2001). Die ermittelten V-Werte der Bewertung sind in Tabelle 1 aufgelistet. Dabei ist der im Vergleich zwischen beiden FIS bessere (also kleinere) Wert weiß hinterlegt, der schlechtere grau. Dadurch zeigt sich deutlich, dass bei nahezu allen Maßen beim Audi MMI deutlich konsistentere Werte über alle Szenen berechnet wurden. Dieses Ergebnis deckt sich mit anderen durchgeführten Untersuchungen, s. z.B. SirValUse (2004).

Tabelle 1: Ergebnisse der Guideline-Bewertung zweier FIS in *Reviser*

	Sender		Navigation		Klang		Über alle Aufgaben	
	Audi	BMW	Audi	BMW	Audi	BMW	Audi	BMW
Column-All Ratio	25,23	35,36	11,83	38,76	86,60	29,81	48,28	35,61
Column Count	0,00	41,42	35,36	47,14	86,60	62,98	55,49	61,81
Text Widget Count	24,49	24,38	18,00	34,68	15,28	30,12	21,59	33,31
Non Widget Area	5,20	1,62	20,14	1,07	2,71	1,37	16,63	1,30
Word Count	29,51	26,26	12,76	36,84	18,62	28,20	27,50	37,05
Gridedness	5,89	13,51	5,74	4,84	1,95	15,80	5,45	20,84
Foreground Color Count	10,53	19,58	11,11	20,00	20,38	27,24	18,71	26,21
Margin East	0,00	155,56	0,00	85,27	16,64	158,77	19,66	154,91
Margin West	0,00	155,56	0,00	111,00	0,00	158,77	0,00	133,16
Margin South	0,00	25,86	0,00	16,26	0,00	21,32	0,00	20,03
Margin North	0,00	26,76	0,00	18,22	0,00	22,04	0,00	21,53
Vertical Area-Balance	31,72	195,98	46,47	1505,70	119,15	258,37	86,68	603,68
Horizontal Area-Balance	40,71	1932,99	46,03	94,16	73,26	297,65	58,45	158,85
Widget Density	16,87	23,47	17,62	28,60	34,64	62,47	33,21	59,70
Widget Count	16,87	23,47	17,62	28,60	34,64	62,47	33,21	59,70

5 Zusammenfassung und Ausblick

Die Entwicklung und Bewertung interaktiver Geräte erfolgt heutzutage aufgrund immer schnellerer Entwicklungszeiten überwiegend werkzeuggestützt. Das ermöglicht den Entwickler, häufig wiederkehrende Arbeiten zu automatisieren und so seine Arbeit zu optimieren.

Nach einer detaillierten Einführung in die Phasen der Systementwicklung wurden die Bereiche *Systementwicklung* und *Systembewertung* auf Unterstützbarkeit von Werkzeugen hin untersucht.

Für die Anwendungsentwicklung wurde beispielhaft das Werkzeug *Impresario* vorgestellt, das vor allem in der Realisierungsphase und Integrationsphase bei der Entwicklung signalverarbeitender Systeme einsetzbar ist und eine Steigerung des Entwicklungskomforts in den genannten Phasen erzielt. *Impresario* steht kostenlos im Internet unter http://www.techinfo.rwth-aachen.de/Software/Impresario zur Verfügung.

Bei der Bewertung interaktiver Systeme gehören Guidelines heutzutage zum Standard in nahezu allen Bereichen des Entwicklungsprozesses. Um eine Automatisierung zu gewährleisten wurde das Werkzeug *Reviser* vorgestellt. Die offene Architektur ermöglicht den Import einer Beschreibung des System-UI unabhängig vom verwendeten Format. Ein Regeleditor erlaubt die komfortable Eingabe und Verwaltung von Guidelines. Die Überprüfung der Systemdaten auf Guideline-Konformität übernimmt ein Expertensystem.

Der entwicklungsbegleitende Einsatz sowie die komfortable Bedienung von *Reviser* stellt eine ernorme Erleichterung der Entwicklungs- und Bewertungsarbeit von Systementwicklern dar.

Literatur

i-Logix Inc. (1997). *Statemate Magnum Reference Manual, Version 1.2*. Andover: i-Logix Inc.

Apple Computer Inc. (2005). *Human Interface Guidelines*. http://developer.apple.com /documentation/UserExperience/Conceptual/OSXHIGuidelines/

Beirekdar, A., Vanderdonckt, J. & Noirhomme-Fraiture, M. (2002). A Framework and Language for Usability Automatic Evaluation of Websites by static Analysis of HTML Sourcecode. In: *4th International Conference on Computer-Aided Design of User*, S. 337–348. Dordrecht: Kluwer Academics Publishers.

Beirekdar, A., Keite, M., Noirhomme, M., Randolet, F., Mariage, C. & Vanderdonckt, J. (2005). Flexible Reporting for Automated Usability and Accessibility Evaluation of Web Sites. In: Costabile, M. F. & Paterno, F. (Hrsg.): *Tenth IFIP TC13 International Conference on Human-Computer Interaction, Lecture Notes in Computer Science.* Berlin: Springer Verlag.

Bleich, H. (2005). Web-Fehler. *c't - Magazin für Computertechnik* (9), 92-93

Booch, G., Jacobson, I. & Rumbaugh, J. (2002). *Professionelle Softwareentwicklung – Das UML-Benutzerhandbuch*. München: Addison-Wesley.

Cantu, M. (2003). *Mastering Delphi 7*. San Francisco: Sybex Inc.

Ellsberger, J., Hogrefe, D. & Sarma, A. (1997). *SDL – Formal Object-oriented Language for Communication Systems.* Hertfordshire: Prentice Hall Europe.

Fels, F. & Hausotter A. (2003). *Moderne Web-Architektur als Kooperationsergebnis*. In *Spectrum*. FHH, Hannover

Hamacher, N., Kraiss, K.-F. & Marrenbach J. (2002). Einsatz formaler Methoden zur Evaluierung der Gebrauchsfähigkeit interaktiver Geräte. *it + ti Informationstechnik und Technische Informatik*, 44 (1), 49-55, Oldenbourg

Hamacher, N. & Kraiss K.-F. (2004). *Expertensystem zur kriterienorientierten Bewertung der Gebrauchsfähigkeit von Dialogsystemen.* In: VDE-Kongress - Innovationen für Menschen. (1, S. 193-198). Berlin: VDE Verlag

Harel, D. (1987). Statecharts: AVisual Formalism for Complex Systems. *Science of Computer Programming, 8*, 231 – 274

Hoffmann, E. R. (1997). Strength of component principles determining direction of turn stereotypes-linear displays with rotary controls. *Ergonomics*, 40 (2), 199-222.

Ivory, M. Y. (2001). *An Empirical Foundation for Automated Web Interface Evaluation.* Dissertation, Computer Science Division, UC Berkeley.

Microsoft DirectX (1995). http://www.microsoft.com/windows/directx

Pfaff, G. E. (Hrsg.) (1983). *User Interface Management Systems: Proceedings of the Seeheim Workshop*. Berlin: Springer Verlag.

Kraiss, K.-F. (1995). *Modellierung von Mensch-Maschine Systemen.* In H. Willumeit & H. Kolrep (Hrsg), *ZMMS-Spektrum, Band 1, Verläßlichkeit von Mensch-Maschine-Systemen* (S. 15-35), Pro Universitate Verlag

Kraiss, K.-F. (Hrsg.) (2006). *Advanced Man Machine Interaction.* Berlin: Springer Verlag (in Druck).

Libuda L., Kraiss K.-F. (2004). Identification of Natural Landmarks for Vision Based Navigation. In *Proceedings of the International Conference on Mechatronics & Robotics 2004 , Aachen, 13.09. – 15.09. 04*, Volume III, 877-882

Mariage, C., Vanderdonckt, J. & Pribeanu, C. (2005). State of the Art of Web Usability Guidelines. In Proctor, R. W. & Vu, K.-Ph.L. (Hrsg), *The Handbook of Human Factors in Web Design.* Mahwah: Lawrence Erlbaum Associates.

Sears, A. (1995). AIDE: a Step Toward Metric-Based Interface Development Tools. In: Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology, S. 101–110. New York: ACM Press.

SirValUse (2004). Best Practice - Mittelkalsse in der Oberklasse. *Usability-Compass* (1), 2–4. Hamburg: Sirvaluse Consulting.

Souza, F. & Bevan, N. (1990). *The Use of Guidelines in Menu interface Design: Evaluation of a draft standard*. In: IFIP INTERACT'90: Human-Computer Interaction, S. 435–440.

Timpe, K. P., Kolrep, H. & Jürgensohn T. (2002). *Mensch-Maschine-Systemtechnik–Konzepte, Modellierung, Gestaltung, Evaluation*. Düsseldorf: Symposion Publishing GmbH.

US Department of Transportation (1998). *Human Factors Design Guidelines*. http://www.fhwa.dot.gov/tfhrc/safety/pubs/atis/

V-Modell (2005). http://www.kbst.bund.de/V-Modell/-,293/V-Modell-XT.htm

Autoren

Dipl.-Ing. Lars. Libuda
Dipl.-Inform. Nico Hamacher

Lehrstuhl für Technische Informatik
RWTH Aachen