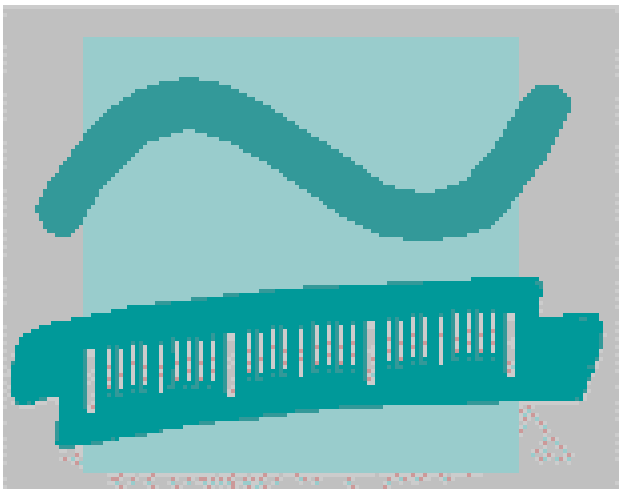
Ein Framework zur Ausnahmebehandlung in mehrschichtigen Softwaresystemen

von

Christoph Knabe

www.tfh-berlin.de/~knabe



Technische Fachhochschule Berlin

Inhalt

1. Motivation
2. Qualitätsziele
3. Diagnosekonzepte
4. Benutzung des Frameworks
   - Erfassung der Diagnoseinfos
   - Ausnahmen melden
5. Realisierbarkeit dieses Frameworks in Java, C++, Ada
6. Erfahrungen / Ausblick

1. Motivation

Praxis-Problem:

**Word kann diese Datei weder speichern noch erstellen. Eventuell ist der Datenträger schreibgeschützt.
(D:\APPLEXC.WW6)**

Hilfetext: Ca. 20 mögliche Ursachen

- **Datenträger schreibgeschützt**
- **Datenträger voll**
- **Datenträger defekt**
- **Zu viele Fenster offen**
- **...**

Lösung: keine

Bewertung: Miserable Diagnoseverwaltung

Vortragsinhalt:

Wie sieht gute Fehlerbehandlung aus?

2. Qualitätsziele

Software-Produkt

- **Fehlertoleranz**
- **Selbsterklärung im Fehlerfall (Diagnosestärke)**

Entwicklungsprozeß

- **Programming by contract (Arbeitsteilung)**
- **Bequemlichkeit**

Qualitätsziel	Erreichbar durch
Fehlertoleranz	**Automatischer Abbruch bei unbehandelter Ausnahme (ab Ada'83)**

programming by contract	**Dienst spezifiziert seine Ausnahmefälle (ab Eiffel'88)**
	`String readLine()`
	`throws EndOfFile`

Diagnosestärke	**MulTEx: Multi-Tier Exception Handling Framework**

3. Diagnosekonzepte in MulTEx

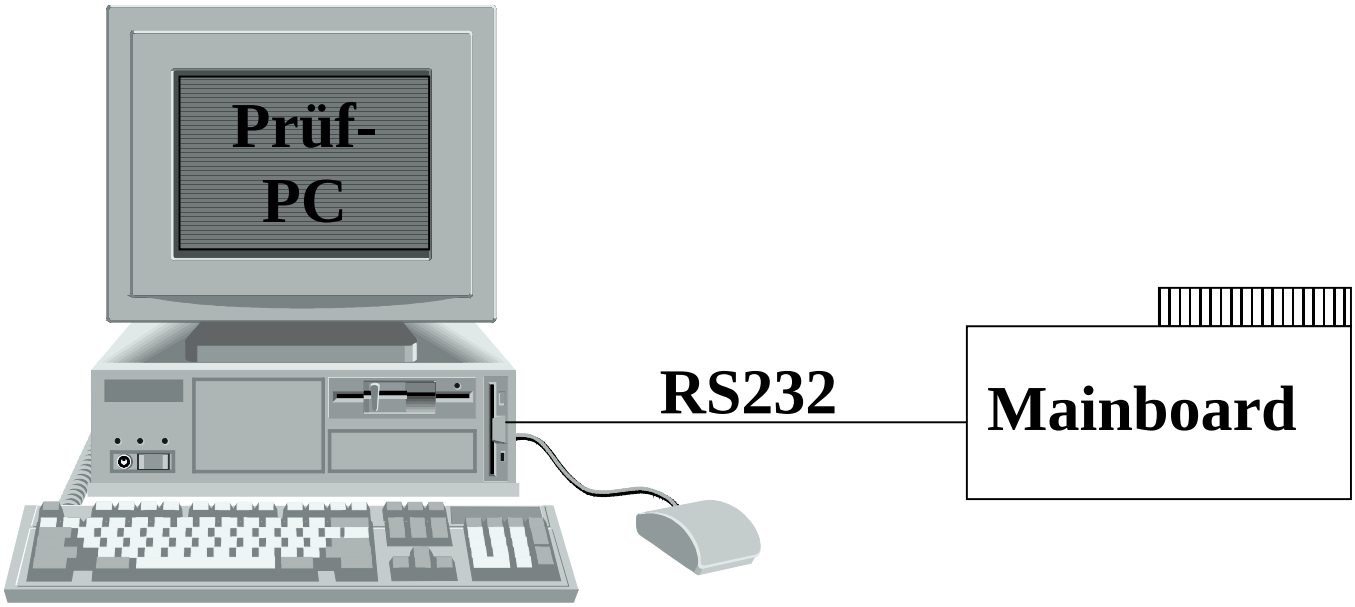
3.1 Ursachenkette

SW-Architektur:

Benutzeroberfläche	`b`
Funktionalität	`f`
Datenhaltung + Dienste	`x`

Anwendung: Prüfprogramm für Mainboards

kommuniziert über Serielle Schnittstelle



Verbindungsaufnahme

Menüpunkt `connect` → **`f`-Schicht**

`f`-Ausnahme `ConnectFailed` ⇒

`b`-Meldung

Cannot connect to the monitor mainboard to be tested

für die Fehlerlokalisierung absolut unzureichend

noch 3.1 Ursachenkette

Mögliche Ursachen

- **Serielle Schnittstelle inexistent / falsch konfigur.**
- **Fehler beim Senden der Initialisier.-Botschaft**
- **Fehler beim Empfangen der Botschaftsquittung**
- **Ressourcenmangel**

Fehlerursache in unteren Schichten bekannt, muß erfaßt und gemeldet werden!

Bsp.: Serielle Schnittstelle inexistent:

Sch	Operation	Schichtadäquate Ausnahme
b	handleConnect ↓	*keine (Meldungsausgabe)*
f	connect ↓	ConnectFailed ↑
x	open ↓	OpenFailed ↑
java	getPortIdentifier	NoSuchPortException ↑

Ursachenkette: Kette der verursachenden Ausnahmen erfassen und mit melden.

Strategie für alle indirekt verursachten Ausnahmen

ansonsten nur vereinzelt: java.rmi.RemoteException

3.2 Weitere Diagnoseinformationen

Stack-Trace

unverzichtbar zur Fehlerlokalisierung
Ortsangaben der Aufrufhierarchie jeweils:

- **Klassenname**
- **Methodenname**
- **Quelldateiname**
- **Zeilennummer**

Ausnahmeparameter, Bsp.:

- **Name des SerialPort**
- **Kommunikationseinstellungen (baud, Bitzahl...)**

Meldungstextverknüpfung

- **Ausnahmen der oberen Schichten mit Meldungstext versehen**
- **Internationalisierbare Texte, Parameterreihenfolge und –formate**

4. Benutzung des Frameworks MulTEx

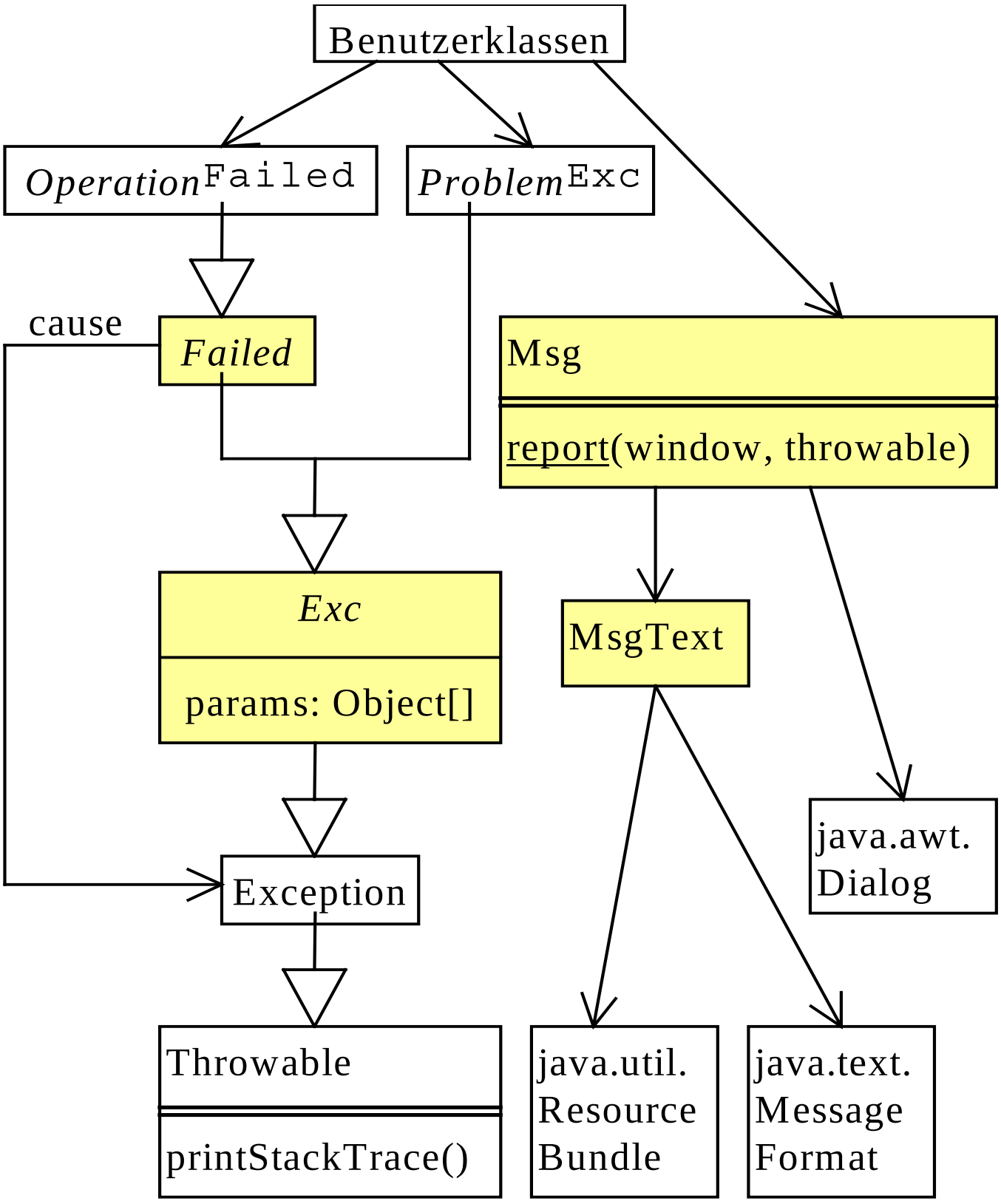


Fig. 1: Das Framework in seinem Kontext

4.1 Erfassung von Ursache und Parametern einer Ausnahme

Ausnahme deklarieren

Konstruktor

Failed(Exception cause, Object[ ] params)

Bsp. davon abgeleitet: x.SerialPort.OpenFailed:

```
class OpenFailed extends Failed {
  public OpenFailed(
    Exception cause, String portName,
    int baudRate, int databits, int stopbits, int parity
  ){
    super(cause, new Object[ ]{ portName,
      ""+baudRate+','+databits+','+stopbits+','+parity
    });
  }
}
```

noch 4.1 Erfassung von Ursache und Parametern

Ursache erfassen und Ausnahme auslösen

Operation x.SerialPort.open kann versagen mit

- **NameExc (originär festgestellt)**
  bei falschem Portnamen
- **OpenFailed (indirekt verursacht)**
  bei von unten kommenden Ausnahmen:
  NoSuchPortException,
  PortInUseException,
  UnsupportedCommOperationException,
  IOException

noch 4.1 Code zur Erfassung von Ursache und P.

```
public void open( String portName,
  int baudRate, int databits, int stopbits, int parity
) throws NameExc, OpenFailed
{ if(!portName.startsWith("COM")){
    throw new NameExc(portName);
  }
try {
  this.portName = portName;
  final javax.comm.CommPortIdentifier portId
  = CommPortIdentifier.getPortIdentifier(portName);
    //NoSuchPortException
  sp = (javax.comm.SerialPort)portId.open(null,0);
    //PortInUseException
  sp.setSerialPortParams(
    baudRate, databits, stopbits, parity
  ); //UnsupportedCommOperationException
  os = sp.getOutputStream(); //IOException
  is = sp.getInputStream();  //IOException
} catch (Exception e) {
  ……  //free resources
  //redefine exception:
  throw new OpenFailed(e,
    portName, baudRate, databits, stopbits, parity
  );
}} //catch,open
```

noch 4.1 Erfassung von Ursache und Parametern

In f/x-Schichten typischer Operationsrumpf:

```
if(Vorbedingung nicht erfüllt){
  throw new ProblemExc(parameter ...);
}
...
try {
  Eigentlicher Algorithmus mit Aufruf von Diensten
} catch(Exception e){
  throw new OperationFailed (e, parameter … );
}
```

4.2 Internationalisierte Meldungstexte

Werden in ResourceBundle-Datei definiert, z. B. in

MsgText_en.properties:

```
x.SerialPort$OpenFailed = Cannot open the \
    serial port "{0}" with communication \
    parameters "{1}"
```

Benutzt: java.text.MessageFormat

4.3 Arbeitsteilung und Benutzeroberfläche

Vorgehen: Erkannte Fehler als abfangbare Ausnahmen auslösen, erst in Oberflächenschicht in Meldung umwandeln.

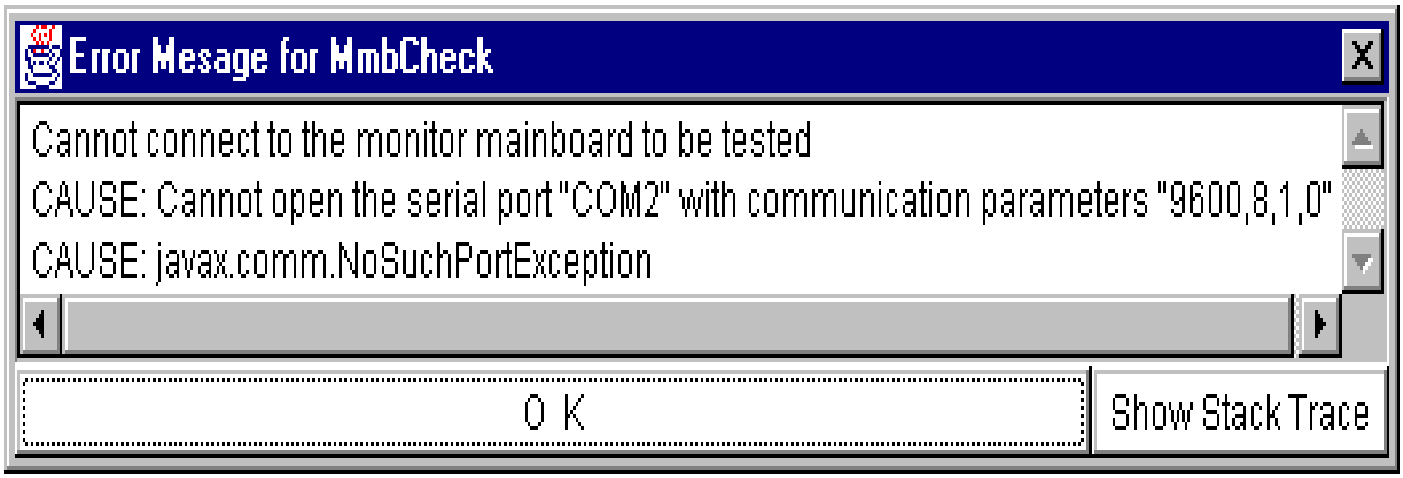
Bsp.: void connect() throws ConnectFailed

Meldungstext: Cannot connect to the monitor mainboard to be tested

In Oberflächenschicht:

```
try { Aufruf einer Operation der Funktionalitätsschicht;
} catch(Exception e) {
   Msg.report(ownerWindow, e);
}
```

Bsp.-Meldungsausgabe mit Ursachenkette:



noch 4.3 Arbeitsteilung und Benutzeroberfläche

Meldungsausgabe mit Ursachenkette:

- Cannot connect to the monitor mainboard to be tested
- CAUSE: Cannot open the serial port "COM2" with communication parameters "9600,8,1,0"
- CAUSE: javax.comm.NoSuchPortException

Bewertung:

- *Verständlich*, da oberste Zeile das Wesentliche enthält
- *Diagnosestark*, da die Informationen der niedrigeren Schichten enthalten sind
- *Bequem* für Programmierer, da ohne Aufwand eine Benutzermeldung mit verschiedenen Ursachenmeldungen kombiniert wird.

4.4 Stack-Trace und Ursachenkette

Fehlerlokalisierung:

Button „Show Stack Trace“ meldet:

- **Unverfälschte Ausnahmenamen, -parameter**
- **Aufruforte rückwärts**
- **„WAS CAUSING:“ markiert Ausnahmenverursachung**

```
javax.comm.NoSuchPortException
 at javax.comm.CommPortIdentifier.getPortIdentifier
  (CommPortIdentifier.java:105)
WAS CAUSING:
x.SerialPort$OpenFailed: {0}=COM2  {1}=9600,8,1,0
 at x.SerialPort.open(SerialPort.java:120)
 at x.SerialPort.<init>(SerialPort.java:53)
 at x.SerialPort.<init>(SerialPort.java:34)
WAS CAUSING:
f.MmbCom$ConnectFailed
 at f.MmbCom.connect(MmbCom.java:160)
 at f.MmbCom.<init>(MmbCom.java:36)
 at f.MmbCheck.<init>(MmbCheck.java:31)
 at b.MmbCheck.handleConnect (MmbCheck.java:504)
 at b.MmbCheck.actionPerformed(MmbCheck.java:212)
 at ...
```

5. Realisierbarkeit dieses Frameworks

Notwendiges Feature	Java	C++	Ada
Ausnahme parametrierbar mit Ausnahmen	+	+	- String
Sammel-Handler für alle Ausnahmen möglich	+ Throwable	-	+ others
Ermitteln des Namens einer Ausnahme	+	+ RTTI	+ Ada'95
Zugriff auf den Stack Trace	+	-	-
Spezifikation der Ausnahmen im Operationskopf	+ Pflicht	0 möglich	- un-möglich
Erben parametrierter Konstruktoren	-	-	-

6. Erfahrungen / Ausblick

Bisheriger Einsatz

- **LAR: Monitor-Mainboard-Prüfprogramm**
- **Diplomarbeiten**

Positiv

+ **Strategie zur Ausnahmebehandlung vorgegeben**
+ **Hilfe gegen erzwungene Ausuferung von throws-Klauseln in den oberen Schichten**
+ **Einfache Meldungstextverknüpfung**
+ **Ausführliche Diagnoseinfos im Fehlerfall**

Ausblick

- **Einsatz im Dokumenten-Managementsystem Hype (DaimlerChrysler Forschung, Berlin)**
- **Freigabe als Open Software**