

HCI-Spezifikationsbeispiel

„Hallo, Welt“-Beispiel in der HCI-Spezifikationssprache LUCIA.

Die zur Formulierung unseres HCI-Beispiels verwendete Oberflächenspezifikationsprache hat den Arbeitsnamen „LUCIA Konkreto“ (language for user centered information architecture). Diese Namenswahl unterstreicht die Absicht, Oberflächen mit dieser Sprache möglichst geradlinig und exakt zu beschreiben.

LUCIA ist eine deskriptive, auf die Beschreibung interaktiver Oberflächen ausgerichtete Sprache. Die Syntax ist zum Zweck der Lesbarkeit an umgangssprachliches Englisch angelehnt.

Die LUCIA HCI-Sprache stellt eine Sammlung von oberflächentypischen Objekten bereit, welche durch Eigenschaftsattribute beschrieben und miteinander in Beziehung gesetzt werden.

Die Sprache deckt die Themenbereiche (Spezifikationsperspektiven) Anforderungssicht, Struktursicht, Dialogsicht und Interaktionssicht, sowie die Schnittstellen zu funktionalen Diensten und zur Darstellungsschicht, ab.

Informationsmodell für Oberflächenspezifikationen

Mit dem unten dargestellten Informationsmodell schlagen wir eine Informationsstruktur vor, die dazu geeignet ist, den Informationsgehalt einer Oberflächenspezifikation aufzunehmen.

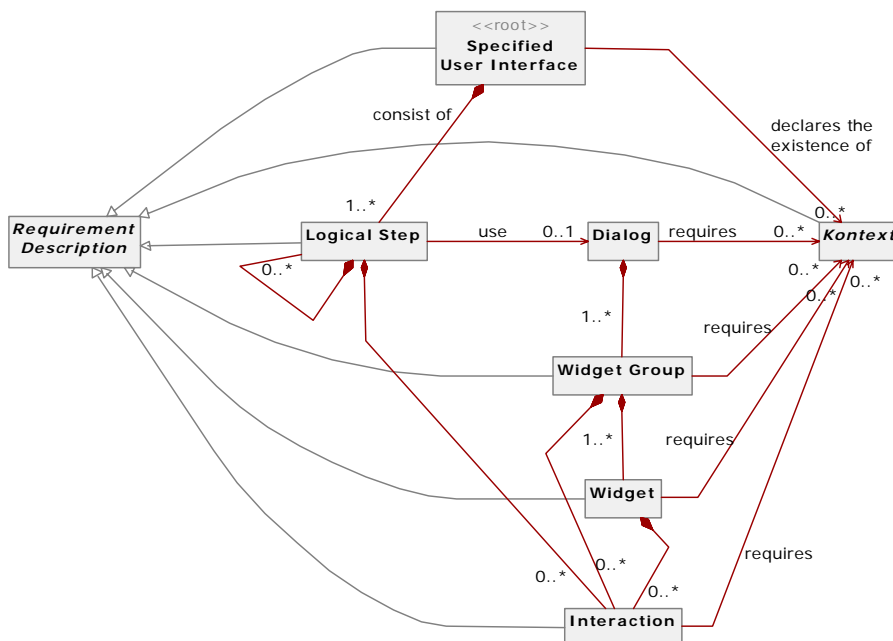


Abbildung 1: Kernentitäten eines HCI-Informationsmodells

Die obige Abbildung zeigt die grundlegenden Elemente einer Benutzeroberfläche und deren Beziehungsgeflecht. Das Informationsmodell klärt zwei zentrale Fragen:

- Welches Informationsgeflecht kann die in einer Oberfläche abzubilden den Informationen wiedergeben?
- Welche Informationen müssen in einer Sprache für Oberflächenspezifikationen formuliert werden können?

Damit bildet das Informationsmodell die Projektionsachse zwischen der Datenstruktur der Laufzeit und der Beschreibungssprache einer Oberfläche.

LUCIA untergliedert die Oberflächenbeschreibung in drei Objekt-Grundtypen:

- **Oberflächen-Objekte**
Logische Bestandteile der Oberfläche, z.B. Ablaufschritte, Bildschirme, Elementgruppen und einzelne IO-Elemente mit den damit verbundenen Interaktionen.
- **Kontext-Objekte**
Logische Platzhalter für die Umwelt der Benutzeroberfläche, z.B. Schnittstellen zum Datenhaushalt und zu den funktionalen Transaktionen der Anwendung, Situationen, Deklarationen von Texten und Grafiken oder Schnittstellen zur grafischen Darstellung logischer Kontrollelemente.
- **Anforderungsobjekte**
Beschreibungen von Anforderungen und Querverweisen.

Die **Syntax** von LUCIA legt fest, wie diese drei Grundobjekte weiter unterteilt werden und nach welchen Regeln sie einander enthalten bzw. aufeinander verweisen können.

Die **Semantik** von LUCIA legt fest, was die Kombinationen von Objekten bedeuten und wie sie als Oberflächenspezifikation zu interpretieren sind.

Zur Abgrenzung möchten wir noch einmal ins Gedächtnis rufen: Konkret ist keine Programmiersprache sondern eine Spezifikationsprache. So legt sie beispielsweise keine Implementierungsdetails fest und toleriert auch unvollständige Konstrukte.

Fangen wir mit einem klassischen „Hallo, Welt“-Beispiel an. Wir wollen eine einfache Benutzeroberfläche beschreiben, die auf dem Bildschirm „Hallo, Welt“ ausgibt und dann mit einem Knopf „Ende“ beendet wird. Anhand dieses trivialen Beispiels wollen wir die wichtigsten Sprachkonzepte und Elemente erklären.

Später erweitern wir unser Beispiel Schritt für Schritt um statische und dynamische Auswahllisten, Interaktion, Situationsabhängigkeit, Funktionsanforderung und Datenabfragen.

hallowelt.hci:

```
HCI-Model named [ Hallo_Welt_Beispiel ]
!! [ Wir wollen eine Benutzeroberfläche beschreiben, die „Hallo, Welt“
ausgibt und dann mit einem Knopf „Ende“ beendet wird. ]
!! named [Anf_2]
[ Das Beispiel soll kurz, einfach und leicht lesbar sein. ]

Start Step named [ Hallo_Welt ] using default Screen

!! [ Im „Hallo, Welt“-Beispiel enthält unsere Oberfläche nur einen
einzigsten logischen Schritt, welcher als IO-Medium den Bildschirm gleichen
Namens nutzt. Der logische Schritt selbst ist in unserem Beispiel
nicht als Kontrollinstrument aktiv. Weitere Einsatzmöglichkeiten der
logischen Schritte werden in anderen Beispielen diskutiert, siehe
~[Anf_2]. ]

!! [ Der Bildschirm dient als Ein- und Ausgabemedium für den logischen
Schritt gleichen Namens, in den er ablaufmäßig eingebettet ist. ]

Screen named [ Hallo_Welt ]
using Layout ~[ Hallo_Anordnung ]
contains
[
Text [ Hallo, Welt! ]
placed at ~[ titelzeile ]
using Widget ~[ spezielles_text_widget ]
Button [ Ende ]
placed at ~[ fusszeile ]
using default Widget
on command unconditionally leave current Step
]
```

```

Layout named [ Hallo_Anordnung ]
!! [ Html ist nur ein Beispiel für die Formulierung eines Layouts. Hier
könnte eine beliebige andere Layoutvorschrift, die der Spezifikationsleser
oder ein Interpreter versteht, stehen. Zum Umfang von Konkreto gehören
lediglich die Platzhalter. ]
[
<html><body><table>
<tr><td>placeholder named [ titelzeile ]</td></tr>
<tr><td>placeholder named [ körper ]</td></tr>
<tr><td>placeholder named [ fusszeile ]</td></tr>
</table></body></html>
]

Widget named [ spezielles_text_widget ]
!! [ Hier ist die Schnittstelle zu den Darstellungselementen. Man kann
hier, umgangssprachlich oder formalisiert, z.B. in SVG oder in HTML
spezifizieren, wie das Widget zur Darstellung des logischen Bedie-
nungselements aussehen soll. ]
[
<h1>~[ value ]</h1>
]

```

Sehen wir uns das Beispiel näher an. Es zeigt, wie die Verquickung von Abläufen, Inhalten, Layout, Darstellung, Interaktionen und Anforderungen systematisch aufgelöst und zu einem abgestimmten Ganzen zusammengefügt wird.

Nachfolgend skizzieren wir, anhand der im „Hallo, Welt“-Beispiel verwendeten Ausdrucksformen, die von Konkreto gelieferten Lösungen zur Formulierung von komplexen Zusammenhängen und Abhängigkeiten zwischen den unterschiedlichen Bausteinen der Oberfläche.

HCI-Model named [Hallo_Welt_Beispiel]

sagt aus, dass unser Spezifikationsmodell den Namen „Hallo_Welt_Beispiel“ hat. Namensdefinitionen werden mit dem Wort „**named**“ eingeleitet. Statt der Anführungszeichen, die wir oft in Fließtexten der Spezifikation benötigen, werden eckige Klammern [] als Begrenzer für Namen und Inhalte verwendet.

In einer Spezifikation kommen viele Namen und Querverweise vor. Deshalb wird zur besseren Unterscheidung eine Namensfestlegung mit „**named [name]**“ notiert und eine Namensreferenz mit „~[**name**]“. Die in einer Konkreto-Spezifikation verwendeten Namen dürfen beliebige Zeichen außer eckigen Klammern, Punkt, Komma und Anführungszeichen ([] . , „“), also auch Leerzeichen und Umlaute, enthalten.

!! [Wir wollen ...]

legt eine Anforderung fest.

!! leitet eine Anforderungsbeschreibung ein. Eine Anforderung kann, muss aber nicht, einen sie identifizierenden Namen haben. „**!! named [Anf_2] [Das Beispiel soll ...]**“ legt eine Anforderung mit dem Namen „Anf_2“ fest. Die Beschreibung der Anforderung folgt in einem ebenfalls mit [] begrenzten Block. Innerhalb der Anforderungsbeschreibung darf mit „~[**name**]“ auf andere Anforderungen und auf beliebige andere benannte Elemente der Spezifikation verwiesen werden.

Eine Anforderung gehört zu dem nächsten, über ihr befindlichen Modellelement. Im „Hallo, Welt“-Beispiel beziehen sich die ersten beiden Anforderungen also auf das Modellelement **HCI-Model** und damit auf die gesamte Oberflächenspezifikation. Die unbenannte Anforderung „**!! [Im „Hallo, Welt“-Beispiel enthält ...]**“ bezieht sich auf „**Step named [Hallo_Welt]**“.

Eine Anforderung muss keinen Namen haben. Allerdings kann dann auch von anderen Stellen der Spezifikation oder auch bei Auswertungen kein Bezug auf diese Anforderung genommen werden. Daher empfiehlt es sich, „echte“ Anforderungen immer mit einem Namen zu versehen.

Eine Anforderungsbeschreibung kann bei Bedarf weiter strukturiert werden, z.B. in Ausgangssituation, Zielsetzung oder Vor- und Nachbedingungen. Wann

und wie diese Möglichkeit angewandt wird, zeigen wir in einem späteren Beispiel, siehe ~[Anf_2] :-).



Start Step named [Hallo_Welt] using default Screen

definiert den ersten Ablaufschritt im Workflow der Benutzeroberfläche. Die Wortfolge „**Start Step**“ zeichnet den Schritt als den ersten Schritt im logischen Ablauf der Oberfläche aus.

Da unser Beispiel nur einen logischen Ablaufschritt hat, würde das Weglassen von „**Start**“ keine Auswirkungen haben. Wenn „**Start Step**“ nicht angegeben wird gilt die Konvention, dass der Ablauf mit dem in der Definitionsreihenfolge ersten Schritt des Modells beginnt.

Höchstens ein **Step** des Modells darf mit dem Prädikat **Start** versehen werden. Typischerweise ist dieser Ablaufschritt eine Login-Maske oder ein Hauptmenü. Der Startschritt übernimmt das Hochfahren der Oberfläche und kontrolliert den anfänglichen Übergang zu anderen Ablaufschritten.

Mit „**using default Screen**“ wird festgelegt, dass der logische Schritt „**Hallo_Welt**“ als Darstellungs- und Eingabemedium einen Bildschirm gleichen Namens nutzt.

Schritte können hierarchisch angeordnet werden. Sie bieten vielfältige Konstrukte zur Ablaufspezifikation mit Unterstützung der orthogonalen Verwendung von hierarchischer und prototypischer Vererbung, die unser einfaches Beispiel jedoch nicht demonstriert.

Screen named [Hallo_Welt]

beschreibt nun die Inhalte der Bildschirmmaske, die der Anwender sieht, und die Interaktionsmöglichkeiten auf dieser Bildschirmmaske.

Mit „**using Layout ~[Hallo_Anordnung]**“ wird dem Bildschirm ein Anordnungsschema zugeordnet, auf das die Dialogelemente Bezug nehmen können.

Innerhalb von „**contains [...]**“ werden die Inhalte des Screens, also Ein- und Ausgabeelemente mit ihren Eigenschaften, beschrieben.

„**Text**“ und „**Button**“ sind logische Dialogelemente. Sie sind auf dieser Ebene noch unabhängig von ihrer tatsächlichen Darstellung (Aussehen) auf der Oberfläche, legen aber fest, was und wie zwischen Anwender und Anwendung kommuniziert wird. Weitere typische logische Dialogelemente, auf die in diesem Beispiel nicht näher eingegangen wird sind Selectors (z.B. Listbox, Combobox, Menü, Checkbox, Radiobutton, Slider), Editfields in verschiedenen Spezialisierungen und Kompositionen aus diesen Grundtypen.

Allen logischen Dialogelementen ist gemeinsam, dass sie Informationen zur und von der Darstellungsebene hin- und her transportieren. Der Informationsgehalt ist dabei abhängig vom logischen Typ des Dialogelements. So dient beispielsweise eine Auswahlliste zur Auswahl aus den angebotenen Möglichkeiten; ein Button hingegen zum Aktivieren einer bestimmten Möglichkeit. Der Informationsgehalt ist jedoch unabhängig vom Look-And-Feel auf der Darstellungsebene. So ist es z.B. für die logische Ebene unwesentlich, ob die Auswahl über RadioButtons oder über eine DropDownListbox getroffen wird.

Logische Dialogelemente sind mit einer Reihe von Eigenschaften ausgestattet, mit denen ihnen Ein- und Ausgabeinhalte, Situationen und Interaktionen, auf denen sie operieren, zugewiesen werden. Die Schnittstelle des logischen Dialogelements zur Darstellungsebene wird ebenfalls durch Eigenschaften ausgedrückt.

In unserem „Hallo, Welt“-Beispiel benutzen wir zur Veranschaulichung der Screenbeschreibung zwei logische Dialogelemente (Text und Button) mit wenigen Eigenschaften.

Text [Hallo, Welt!]

placed at ~ [titelzeile]

using Widget ~ [spezielles_text_widget]

legt fest, dass der Text „**Hallo, Welt!**“ an der Layoutposition „**titelzeile**“ ausgegeben werden soll. Zur Darstellung des Texts soll dabei die im Widget „**spezielles_text_widget**“ festgelegte Renderingvorschrift verwendet werden.

Button [Ende]

placed at ~ [fusszeile]

using default Widget

on command unconditionally leave current Step

legt fest, dass der mit „**Ende**“ beschriftete, unbenannte Command Button an der Layoutposition „**fusszeile**“ angezeigt werden soll. Zum Rendering des Buttons soll dabei ein generisches Widget (z.B. der von Windows bereitgestellte Pushbutton) verwendet werden. Wenn der Button geklickt wird, soll der Screen und der Step „**Hallo_Welt**“ und damit die Anwendung verlassen werden. Beim Drücken des Buttons werden keine Bedingungen geprüft und keine Rückfragen (etwa „Sind Sie sicher?“) gestellt.

Die vielfältigen Möglichkeiten der Situationsprüfung und der Reaktion auf Eingaben und Kommandos werden wir anhand von anderen Beispielen behandeln.

Layout named [Hallo_Anordnung] [...]

leitet die Definition eines Layouts ein. Das Layout beschreibt eine Schnittstelle zur Darstellungsebene der Oberfläche. Im Layout können die statischen Anteile von Screens beschrieben und der Screen in Bereiche unterteilt werden.

Die Beschreibung des Layouts kann in einem beliebigen Format erfolgen und ist von Konkreto nicht definiert. Zur Konkreto-Sprache gehören lediglich die Platzhalter, auf die sich Dialogelemente zur Positionierung mit „**placed at** ~ [**platzhaltername**]“ beziehen können. Ein Positionierungsplatzhalter wird mit „**placeholder named [platzhaltername]**“ festgelegt.

Zur Festlegung von Layouts bieten sich vor allem HTML und SVG an. Zwischen Auszeichnungselementen können die Platzhalter eingestreut werden, an welchen die Dialogelemente eingesetzt werden können. Dialogelemente, die keinen Bezug auf Layoutplatzhalter nehmen, landen im Bereich „**placeholder named [content]**“, wenn das Layout einen solchen Platzhalter festlegt.

Benutzt der Screen kein Layout, dann werden die Elemente in der Definitionsreihenfolge platziert. Hat das vom Screen benutzte Layout keinen [**content**] Platzhalter, dann werden die Dialogelemente nach den Layoutelementen auf dem Bildschirm ausgegeben.

Zur Festlegung von Widgets bieten sich, wie schon für Layouts, HTML und SVG an. Zwischen den Auszeichnungselementen, die das Widget rendern, können die Platzhalter eingestreut werden, an welchen die Eigenschaften von Dialogelementen eingesetzt werden können.

Widget named [spezielles_text_widget] [...]

leitet die Definition eines Widgets ein. Das Widget beschreibt wie das Layout eine Schnittstelle zur Darstellungsebene der Oberfläche. Im Widget kann das Aussehen und die interne Präsentationslogik (z.B. wie sich das Widget beim Fokussieren verändert) des Darstellungselements beschrieben werden.

Die Beschreibung des Widgets kann in einem beliebigen Format erfolgen und ist von Konkreto nicht definiert. Zur Konkreto-Sprache gehören lediglich die Platzhalter für den Inhalt „**[value]**“ und weitere Eigenschaftswerte von logischen Dialogelementen, auf die unser Beispiel jedoch nicht eingeht.

Unser Beispiel zeigt nur eine kleine Auswahl der Sprachkonstrukte der Oberflächenspezifikationsprache Konkreto. Es sollte uns jedoch in Form eines vertikalen Durchstichs einen Eindruck von den unterschiedlichen Themen (Sichten) verschaffen, die in einer Spezifikation beschrieben und in Bezug zueinander gesetzt werden müssen.

pch, 08/2004
last revised 01/2006

Paul Chlebek
HCI • GUI • UML • OOD • MDA • XSLT

mailto: pc@benutzeroberflaeche.de
phone: +49 173 106 1830

<http://www.projectpeople.net/chlebek/>
<http://www.benutzeroberflaeche.de>