Pattern-Sprachen und Automatisierung der GUI-Entwicklung

Alexander Roski und Christian Märtin

Fachhochschule Augsburg, University of Applied Sciences
Fakultät für Informatik
Baumgartnerstr. 16
D-86161 Augsburg
alex_roski@yahoo.de
maertin@informatik.fh-augsburg.de

Abstract: Dieses Paper erläutert einen prototypischen Ansatz für die automatisierte Verwendung strukturierter Pattern-Sprachen als Entwurfs-Hilfsmittel zur Erstellung grafischer Benutzungsschnittstellen. Im Einzelnen werden die dazu erforderlichen Rahmenbedingungen betrachtet, der Entwicklungs-Prozess erklärt und die einzelnen Sprachelemente vorgestellt. Abschließend erfolgt eine reflektorische Betrachtung, inwieweit dieser Ansatz die in ihn gesetzten Erwartungen bereits erfüllt und welche Frage- oder Problemstellungen in zukünftigen Betrachtungen und mit fortschreitender Entwicklung noch auftreten können.

1 Einführung

Wirtschaftlichkeit, kurze Entwicklungszyklen, Flexibilität, Effizienz, Einfachheit: das alles sind Eigenschaften, die ein Produkt heutzutage mindestens vorweisen sollte. Höchste Qualität in kürzester Zeit zu erreichen ist gegenwärtig *der* Leitsatz, der in immer mehr Entwicklungsprojekten zur Anwendung kommt. Da sich diese Argumentation im globalen Wettbewerbs-Umfeld leicht rechtfertigen lässt, gewinnen Ansätze, die nach dieser Devise arbeiten, immer mehr an Bedeutung. Computer ermöglichen es heute, in vielen industriellen Entwicklungsprojekten den Zeitfaktor durch Automatisierung zu reduzieren. Kommerzielle Generierungs-Umgebungen wie OlivaNova [He99] zeigen aber auch, dass sich Automatisierung bei der Konstruktion interaktiver Softwaresysteme produktiv einsetzen lässt.

Durch die Einführung von Pattern-Sprachen [AIS77] erhielten die Entwickler ein weiteres Abstraktions-Konzept, das sie in die Lage versetzte, an der Stellschraube Qualität zu drehen. Strukturales, organisatorisches und inhaltliches Wissen, das über Jahre hinweg erarbeitet wurde, eröffnet in Kombination mit der automatisierten Verarbeitung ungeahnte Möglichkeiten, den Entwicklungsprozess zu verbessern. Die Idee, dass es ausreichen könnte, nur das Wissen akquirieren und speichern zu müssen, um daraus direkt die Produkte ableiten zu können, dürfte jeden Wissenschaftler faszinieren.

Es stellt sich aber die Frage, ob Pattern-Sprachen, hierfür wirklich das richtige Konzept sind. Die Ausbeutung von Patterns für das modellbasierte Engineering wird in [SGR04] ausführlich diskutiert. In unserem Ansatz einer automatisiert auslesbaren Sprache in Form von Patterns nach [AIS77] soll ein Versuch unternommen werden, diese Idee für die Entwicklung von grafischer Benutzungsschnittstellen anzuwenden [MR07].

2 Ziele und Rahmenbedingungen

Das Ziel des hier vorgestellten Ansatzes ist es, einen Entwurf zu liefern, mit dessen Hilfe ein Softwareentwickler in kürzester Zeit grafische Programmoberflächen mit hoher Usability erstellen kann. Das Wissen, das in der verwendeten Pattern-Sprache vorliegt, soll dazu verwendet werden zu definieren, wie die Usability-Ziele erreicht werden können. Konkret soll die Sprache Informationen zur adäquaten Gruppierung und Positionierung der Elemente enthalten. Grundsätzlich muss dem Entwickler dabei bekannt sein, über welche Grundstruktur seine Applikation verfügt, also gewissermaßen auch, welche Elemente in den jeweiligen Programmteilen enthalten sind und welche Aufgabenstruktur die jeweiligen Programmteile besitzen. Die Pattern-Sprache soll hierfür zwar Unterstützung bieten, doch sind aktuelle Anforderungen an Programme so unterschiedlich, dass es nahezu unmöglich erscheint, alle aktuellen und zukünftigen Eventualitäten abdecken zu können. Es bietet sich aber an, in der Sprache auch allgemein gehaltene Default-Patterns zu hinterlegen, um die Auswirkungen unvorhergesehener Anforderungen abzumildern.

3 Entwicklungsprozess

Für den Prozess zur Erstellung der grafischen Benutzungsschnittstelle, werden zunächst zwei unterschiedliche Phasen definiert. Die erste Phase dient zur Konstruktion eines semi-abstrakten Modells der Schnittstelle. Die zweite Phase ist für die Umsetzung des Modells in ein konkretes Ergebnis zuständig. Zunächst geht es also darum, den Entwickler durch die Pattern-Sprache zu führen, notwendige Entscheidungen treffen zu lassen und dabei das entsprechende semi-abstrakte Modell der finalen Benutzer-schnittstelle zu kreieren. In der zweiten Phase werden aus dem Modell automatisch die Objektklassen generiert, die anschließend in das eigentliche Software-Projekt integriert werden können. Wird als Entwicklungssprache z.B. Java verwendet, bedeutet das im konkreten Fall, dass Java-Dateien als Ergebnis erzeugt werden. Diese lassen sich dann in die gewählte Projektumgebung importieren und mit den Logikkomponenten verbinden. Anschließend kann die gesamte Software erstellt werden.

Der Grund für einen zweistufigen Entwurf liegt zum einen darin, dass Abhängigkeiten von Entscheidungen seitens des Entwicklers vorliegen können, die sich eventuell auf bereits erstellte Klassen auswirken würden. Zum anderen sprechen aber auch Argumente wie mögliche Änderungswünsche, einfache Speicherung oder schlichte Wiederverwendung zunächst für eine Darstellung der Oberfläche als semi-abstraktes Modell. Somit ergeben sich auch für die Sprachelemente zwei unterschiedliche, aber zusammenhängende Typen von Konstrukten, die im Folgenden genauer erklärt werden sollen.

4 Sprachelemente

Da die Entwicklung bzw. Weiterentwicklung der Pattern-Sprache so einfach wie möglich sein soll und die Implementierung einer Generator-Applikation mit unterschiedlichen Programmiersprachen erfolgen kann, fiel die Wahl aufgrund der hohen Flexibilität, Einfachheit und Systemunabhängigkeit auf XML [W3C].

4.1 Elemente innerhalb der Pattern-Sprache und Modellgenerierung

Die ursprünglichen Patterns sind somit ebenfalls komplett in XML strukturiert und enthalten Phrasen wie Problem, Kontext und Lösung wie in [AIS77] und [Al79] beschrieben. Für die Automatisierung wurde ein Zweig *automation* definiert. Grundsätzlich ähnelt dieser durch seine Funktion einer Datentyp-Definition oder einem Schema [W3C], allerdings werden hier nicht nur die Art und Struktur eines Typs festgelegt, sondern auch bereits Informationen in Form der semi-abstrakten Elemente. Wie aus den Abbildungen 1 und 2 deutlich wird, gibt es zwei Bereiche: *code* enthält Informationen, die für die Erstellung des Modells notwendig sind und *children* Angaben darüber, welche Arten von Kindelementen überhaupt erlaubt sind sowie bereits genaue Daten darüber, mit welchen Informationen diese Kindelemente versorgt werden können.

```
<automation>
  <code>
    <element>panel</element>
    <attribute name="name" obliged="true" />
    <attribute name="type" obliged="true">ByFrame</attribute>
  </code>
  <children elements="3">
    <child id="1" ref="C6" min="0" max="*">gridx=0;gridy=1</child>
  </children>
</automation>
```

Abbildung 1:Ausschnitt des Patterns *ByFrame* B2[1]

Insgesamt soll dieses Beispiel die Struktur eines OK-Abbrechen-Fensters beschreiben. Abb. 1 beschreibt den Aufbau des Fensters, Abb2. den Aufbau des OK_Abbrechen-Patterns. B2 definiert somit ein Element mit dem Name *panel* und den Attributen *name* und *type*. Beide Attribute sind zwingend erforderlich *obliged="true"* und werden entweder vom eigenen Wert, wie hier *ByFrame*, interaktiv vom Entwickler oder vom übergeordneten Pattern angefordert. Die Übergabe von Informationen an Kindelemente ist im Knoten *children* definiert. Jedes Kindelement wird einzeln angegeben (Abb. 1). Mit *ref* wird das Kindelement bestimmt, *min* gibt an, wie oft es mindestens vorkommen darf, *max* wie oft maximal. Wenn *min* = 0 ist, kann der Entwickler entscheiden, ob er dieses Kind mit anlegen möchte oder nicht. Im Wertbereich sind die Übergabeinformationen enthalten, die für die Kind-Elemente bestimmt sind. Hier im Beispiel werden an das Kind mit der *id* 1, das vom Typ *ref* C6 ist, die Werte *gridx* und *gridy* übergeben. Das Pattern in Abbildung 2 entspricht einem Kind vom Typ C6 und verwertet diese Daten.

```
- <automation>
  - <code>
      <element>panel</element>
      <attribute name="name" obliged="true">OK/Cancel</attribute>
      <attribute name="type" obliged="true">internalPanel</attribute>
      <attribute name="gridx" obliged="true" />
      <attribute name="gridy" obliged="true" />
      <attribute name="gridwidth" obliged="false" />
      <attribute name="gridheight" obliged="false" />
      <attribute name="weightx" obliged="false" />
      <attribute name="weighty" obliged="false" />
      <attribute name="fill" obliged="false" />
      <attribute name="anchor" obliged="false" />
      <attribute name="insets" obliged="false" />
      <attribute name="ipadx" obliged="false" />
      <attribute name="ipady" obliged="false" />
    </code>
  - <children elements="2">
      <child id="1" ref="D3" min="1" max="1">gridx=0;gridy=0;fill=NONE;
                                             anchor=EAST;gridwidth=1.0;value=OK</child>
      <child id="2" ref="D3" min="1" max="1">gridx=1;gridy=0;fill=NONE;
                                             anchor=EAST;value=Abbrechen</child>
    </children>
  </automation>
```

Abbildung 2: Ausschnitt des Patterns OK_Cancel C6[1]

C6 definiert hierbei ein Panel mit zwei Buttons. (D3 == Buttons). Hierzu benötigt es zwingend vier Attributangaben: *Name*, *Type*, *Gridx* und *Gridy*. Die ersten beiden sind hinterlegt, die anderen werden vom Eltern-Pattern übergeben (s. Abb. 1). Für die eigenen Kindelemente (zwei Buttons) werden ebenfalls Werte übergeben. Dieser Vorgang wird so lange wiederholt, bis in einem Pattern keine Kindelemente mehr vorkommen. In diesem Beispiel wird das im Pattern *Button* der Fall sein. Der Modell-Generator würde für dieses Beispiel den folgenden Eintrag schreiben.

```
- <windows>
  - <panel name="OK Cancel Frame" type="ByFrame">
    - <panel name="OK_Cancel" type="internalPanel" gridx="0" gridy="1">
        <button gridx="0" gridy="0" fill="NONE" anchor="EAST" gridwidth="1.0">OK</button>
        <button gridx="1" gridy="0" fill="NONE" anchor="EAST">Abbrechen</button>
      </panel>
    </panel>
  </windows>
```

Abbildung 3: Auszug aus einem erstellten Modell

4.2 Elemente des Modells

Das erzeugte Modell ist nun das Ergebnis der ersten Phase und entspricht, wie bereits erwähnt, einer semi-abstrakten Darstellung der zu erstellenden grafischen Schnittstelle. Die hier verwendete Syntax entstammt einer Eigenentwicklung und ist stark an Java angelehnt. Prinzipiell ist aber nicht festgelegt, welche Syntax für ein solches Modell verwendet werden muss. Denkbar sind ebenso Ansätze in XUL [Xul] oder XAML [Xaml]. Abhängigkeiten bestehen hier lediglich zwischen den definierten Elementen in der Pattern-Sprache. Der Aufbau des Modells wird hier nur angedeutet. Im Mittelpunkt dieses Beitrags steht die exemplarische Vorstellung, inwieweit es möglich ist, Pattern-Sprachen mit automatisiert lesbarem Code zu versehen.

Grundlage ist dabei auch die bereits in [MR07] definierte Struktur einer Pattern-Sprache, mit deren Hilfe der beschriebene Durchlauf erst ermöglicht wird.

Die in der zweiten Phase des Entwicklungs-Prozesses noch ausstehende Aufgabe, ist es, dieses Modell einzulesen und in die entsprechenden Klassen-Dateien umzuwandeln. Beispielhaft würde das oben erzeugte Modell nach der Umwandlung in die Java-Dateien und anschließend mit einem Compiler übersetzt, folgende GUI ergeben.

Abbildung 4: Darstellung des Modells nach Übersetzung in Java-Code

5 Resümee

Die generelle Idee von Pattern-Sprachen, nur abstrakte Best-Practice-Lösungen zu bieten, wird zwar vom dargestellten Ansatz teilweise unterlaufen, da aufgrund der Automatisierung auch unbedeutende kleine Patterns geschrieben werden müssen. Doch der Gedanke, ein Regelwerk anzubieten, mit dem sich Software mit höherer Usability entwickeln lässt, bleibt nach wie vor erhalten. Letztendlich zeigen wir mit der hier vorgestellten Vorgehensweise einen funktionsfähigen Ansatz, mit dessen Hilfe sich in kürzester Zeit grafische Benutzungsschnittstellen entwickeln lassen, die die Usability-Anfoderungen in der Form erfüllen, wie sie in der Pattern-Sprache definiert wurden. Prototypisch wird der Ansatz derzeit für die Anwendungsdomäne *Benutzungs-oberflächen für industrielle Steuerungsanwendungen* ausgearbeitet. Ein weiterer Aspekt, der in diesem Zusammenhang noch zu erforschen ist, ist die Einbindung von ausführbaren Programmroutinen in die Pattern-Sprachen.

Literaturverzeichnis

[AIS77] Alexander, C.; Ishikawa, S.; Silverstein, M.: A pattern language, Oxford University Press, 1977

[Al79] Alexander, C.: The Timeless Way of Building, Oxford University Press, 1979

[He99] Hernandez, J.: Model based multi-tier & multi-platform application generation, OlivaNova; Care Technologies S.A., http://www.care-t.com/index.asp

[MR07] Märtin, C., Roski, A.: Structurally Supported Design of HCI Pattern Languages. Erscheint in: Proceedings of HCI International, Beijing, 22-27 July, 2007, Springer LNCS 4450

[MS02] Middendorf, S.; Singer, R.; Heid, J.: Java – Programmierhandbuch und Referenz für die Java-2-Plattform, Standard Edition, dpunkt.verlag, 3-Auflage, 2002

[Se04] Seeboerger-Weichselbaum, M.: Das Einsteigerseminar XML, VMI Buch, 2004

[SGR04] Sinnig, D., Gaffar, A., Reichart, D., Forbrig, P.: Patterns in Model-Based Engineering, Proc. CADUI 2004, Kluwer Academic Publishers, 2005, pp. 197-210

[W3C] XML Extensible Mark-up Language http://www.w3c.org/XML/

[Xaml] XAML, http://msdn2.microsoft.com/en-us/library/ms752059.aspx, 2007

[Xul] XUL, http://www.mozilla.org/projects/xul/, 2007