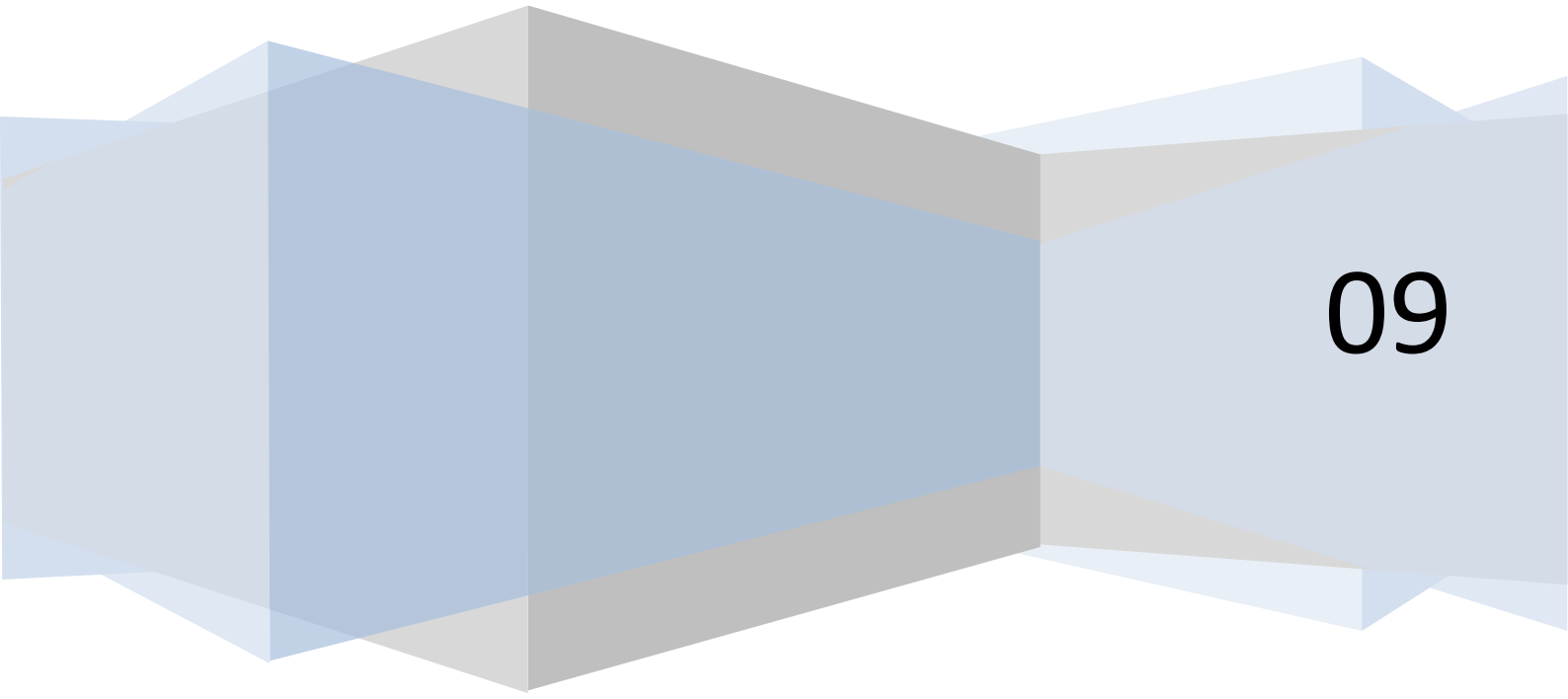
Hochschule für Angewandte Wissenschaften Hamburg

Projektbericht

Sommersemester 2009

Thorben Pergande



09

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung & Motivation

Diese Ausarbeitung stellt den Bericht zur Durchführung des Masterprojektes im Sommersemester 2009 im Themenbereich „Unterstützung des Softwareerstellungsprozesses durch das Microsoft Surface an der Hochschule für Angewandte Wissenschaften Hamburg dar. An diesem Projekt nahmen ebenfalls Daniel Löffelholz und Hauke Wittern unter der Betreuung von Prof. Dr. Olaf Zukunft teil.

Das Gruppenprojekt hatte sich zur Aufgabe gesetzt, Erfahrungen in dem Umgang und der Programmierung des Microsoft Surface zu sammeln. Dabei ging es zunächst darum, die verwendeten Technologien (*XAML, Expression Blend, C#, Surface SDK*) zu erlernen und herauszufinden, welche Möglichkeiten und Einschränkungen bei der Programmierung von Applikationen in diesem Kontext gelten. Dazu wurde als Pilotprojekt ein Framework entwickelt, dass die Darstellung, Bearbeitung und Reviews von beschreibenden Diagrammen (zunächst UML als Schwerpunkt) unterstützt. Hierbei wurden vor allem zu Beginn vorwiegend in der gesamten Gruppe gearbeitet, damit das bestehende Wissen um die Technologien schnell über das gesamte Team verteilt wird, danach hat jeder Teilnehmer eigene Schwerpunkte bei der Erstellung des Frameworks gesetzt, auf die in dieser Ausarbeitung speziell eingegangen wird. Gegen Ende des Projekts wurden mehrere Bewerbungen in Form von Konferenznotizen bei verschiedenen Konferenzen im Multitouch-Umfeld erstellt.

Hauke Wittern befasste sich vorwiegend mit der Entwicklung und Umsetzung der Architektur des Frameworks und einzelnen Funktionen, wie *Drag and Drop* auf Multitouch Systemen. Daniel Löffelholz befasste sich vorwiegend mit der Erstellung der Graphen, dem Design der Funktionsfelder als auch der Speicherung/Ladung ganzer Diagramme mittels XMI. Thorben Pergande erstellte größtenteils die Modelle der UML-Diagramme und die dazugehörigen ViewModels, befasste sich mit der Möglichkeit, echte Objekte mittels Tags auf dem Surface einzusetzen und der geeigneten Darstellung der Informationen auf dem Surface.

Kapitel 2 befasst sich mit den Projektzielen, indem auf die Vision, das Szenario und die Anforderungen eingegangen wird. Kapitel 3 stellt den Projektablauf bei der Erstellung des Frameworks dar. Kapitel 4 zeigt die eigenen Schwerpunkte der Arbeit an dem Projekt auf. In Kapitel 5 wird ein Fazit über das vergangene Projekt (erste Projekthälfte) gestellt und in Ausblick gestellt, was im zweiten Teil des Projekts geplant ist.

2. Projektziele

In diesem Kapitel werden die Projektziele des Masterprojekts am Microsoft Surface im Sommersemester 2009 dargestellt.

2.1 Vision

Die Vision des durchgeführten Projekts bestand darin, die Möglichkeiten des Microsoft Surface für den Einsatz in der Softwareerstellung zu evaluieren und die Konzepte für die Erstellung geeigneter Software dafür zu erlernen. Dazu wurde ein Framework für die Darstellung beliebiger grafenbasierter Diagramme prototypisch erstellt. Grafenbasierte Diagramme wurden daher gewählt, da die meisten bekannten und eingesetzten Modellierungsmethoden auf Kanten und Knoten basieren. Im ersten Schritt soll das Framework Klassen- und Komponentendiagramme aus der UML 2.0 verarbeiten

können. Als weitere Diagrammtypen sind Petri-Netze oder EPK geplant. All diese Diagramme basieren auf Informations-Containern (Knoten) und Beziehungen (Kanten) zueinander. Das erstellte Framework soll Informationen über die verschiedene Darstellung der Knoten und Kanten beinhalten, aber weitestgehend dasselbe funktionale Anwendungsverhalten für die verschiedenen Diagrammtypen beinhalten.

2.2 Szenario

Das angestrebte Szenario für den Einsatz eine Multitouch-Tisches unter Einsatz eine Diagrammframeworks für diverse Diagrammtypen ist das eine Review-Prozesses zwischen Entwickler und Kunden, Team, Vorgesetzten etc. Dabei wird davon ausgegangen, dass ein Diagramm an einem Arbeitsplatz erstellt wird und das Surface für die Präsentation genutzt wird. Da die Technik die Interaktion mit mehreren Anwendern ermöglicht, wird davon ausgegangen, dass ein Iterationsstand des Designs zusammen mit dem Auftraggeber an dem Multitouch-Tisch analysiert wird. Änderungen können dann direkt von allen Beteiligten am System vorgenommen werden. Dies beinhaltet z.B. neue Klassen zu erstellen, bestehende Klassen zu verändern als auch die Beziehungen zwischen Knoten zu ändern/definieren. Solch ein Szenario soll durch die erstelle Lösung des Frameworks auf dem Microsoft Surface realisiert werden.

2.3 Anforderungen

Die Vision und das Szenario sowie die vorgegebene Hardware geben einige Anforderungen an das Framework und darauf basierende Anwendungen vor. Die Anforderungen gelten genauso für das exemplarische UML-Modellierungs-Werkzeug. Im Folgenden werden zunächst die fachlichen und anschließend die technischen Anforderungen vorgestellt.

2.3.1 Fachliche Anforderungen

Die auf dem Framework basierenden Anwendungen müssen die folgenden fachlichen Anforderungen erfüllen:

- Zwei bis fünf Benutzer müssen die Anwendung gleichzeitig bedienen können. Dies ist eine Voraussetzung für das Szenario der kollaborativen Arbeit.
- Die Benutzer müssen alle Funktionen der Anwendung mit den Fingern und mit auf dem Surface befindlichen physikalischen Gegenständen bedienen können. Klassische Eingabemethoden, wie z.B. Maus und Tastatur, darf die Anwendung nicht voraussetzen.

Das Framework soll es erleichtern eine Anwendung zu entwickeln, welche die vorigen Anforderungen erfüllt. Daraus folgen Anforderungen an das Framework und seine Architektur:

- Das Framework muss Grundfunktionalität zum Anzeigen und manipulieren von Modellen bereitstellen. Dazu gehören das Verschieben, Löschen und Hinzufügen von Elementen eines Modells.
- Das Framework muss allgemeine Werkzeuge für das kollaborative Arbeiten mit Modellen bereitstellen.
- Zusätzliche Werkzeuge (z.B. für spezielle Domänen) müssen einfach entwickelt und hinzugefügt werden können.
- Die Architektur des Frameworks muss es ermöglichen auf einfache Weise neue auf dem Framework basierende Anwendungen zu entwickeln.

- Das Framework darf soll möglichst wenig Voraussetzungen an die Domäne der Modelle stellen.

Die Architektur des Frameworks muss es ermöglichen auf einfache Weise die abstrakte und konkrete Syntax einer Domäne zu definieren. Die Anwendungslogik bestehender Domänen (z.B. Implementierungen der UML) muss als abstrakte Syntax der Domäne wiederverwendet werden können. Die konkrete Syntax muss auf einfache Weise beschrieben werden können.

2.3.2 Technische Anforderungen

Die in diesem Projekt verwendete Hardware ist ein Microsoft und ist fest vorgeben. Daraus folgen einige technische Anforderungen an die Implementierung, damit diese die Funktionen der Hardware verwenden kann und den Richtlinien für Surface-Anwendungen *SurfaceUserExperienceGuidelines* entspricht.

- Die Implementierung muss auf Basis des Microsoft .Net Frameworks 3.5 erfolgen.
- Für die Implementierung der grafischen Benutzungsoberfläche müssen die *Windows Presentation Foundation* (WPF) und das auf der WPF basierende Surface SDK 1.0 verwendet werden. Die WPF ist Teil des dotNet Frameworks.
- Als Implementierungssprachen sollen C\# und XAML (*Extensible Application Markup Language*) verwendet werden.

3. Projektablauf

Dieses Kapitel soll beschreiben, wie der Ablauf der Erstellung des Frameworks von statten ging. Dabei kann das Projekt in vier Phasen unterteilt werden: Umgang mit dem Surface erlernen, Ideen für Framework sammeln, Implementation und Evaluierung. Die letzten drei Phasen wurden iterativ durchgeführt mit einer Iterationsdauer von einer Woche.

3.1 Umgang mit dem Surface

Zu Beginn des Projekts fanden wir ein Microsoft Surface inkl. der bereits installierten Demos vor. Der Umgang mit XAML und die Architektur von Microsoft Surface Anwendungen (MVVM, XAML etc.) wurde dann im gesamten Team erlernt, wobei der Wissensstand bei den Teilnehmern zu diesen Themen nicht gleich war. Ein Projektmitglied hatte bereits Erfahrungen auf dem MS Surface gesammelt und dessen Wissen wurde durch die Erstellung mehrerer kleiner Beispielapplikationen mit dem Rest des Teams geteilt. So war die Anlernkurve für die ersten Schritte relativ flach.

3.2 Ideen, Projektziel und erste Schritte

Die bereits in Kapitel 2 als Vision und Szenario beschriebenen Ideen und Ziele wurden im gemeinsamen Kreis definiert. Mit der Basisidee, ein Framework für graphenbasierte Diagramme als Unterstützung im Softwareentwicklungsprozess zu erstellen, wurde zunächst ein sehr einfacher Prototyp erstellt, der eine Klasse darstellen konnte.

3.3 Prototypisches Vorgehen im Projekt

Anhand des ersten Prototyps konnten erste Hindernisse und weitere Anforderungen im Kreis des Projektteams unter Hilfestellung weiterer Kommilitonen (Tester) und Professoren inkl. deren Feedback erkannte und erstellt werden. Aus Basis dieser Informationen und Ideen wurden neue

Teilziele definiert und die Aufgaben innerhalb des Teams aufgeteilt. Jede Woche wurde am Montagmorgen ein Treffen vereinbart, indem der neueste Prototyp vorgestellt und über Probleme, neue Ideen und Anforderungen diskutiert wurde.

4. Projektschwerpunkte

An dieser Stelle werden die von Thorben Pergande gewählten Projektschwerpunkte genauer dargestellt und erläutert. Dies ist nur eine Schwerpunktübersicht, jeder Teilnehmer des Projekts hat auch in den anderen Teilbereichen unterstützt.

4.1 Model-View-ViewModel (MVVM)

Für den Einsatz bei Definitionssprachen wie XAML hat sich eine Änderung des klassischen MVC hin zum sog. MVVM vollzogen. Dabei wird der Controller durch ein ViewModel ersetzt. Der eigentliche Unterschied zwischen Controller aus dem klassischen MVC und dem MVVM z.B. für XAML ist, dass das ViewModel aus MVVM eine Abstraktion für verschiedene Views bildet. Dieser wird dann per Databinding an den View gebunden. Für den Einsatz im Diagramm-Framework wurde zunächst das MVVM-Pattern angewendet, um UML-Klassendiagramme zu definieren. Dazu wurden zunächst die Modelle als Klassen in C# für ein Klassendiagramm erstellt.

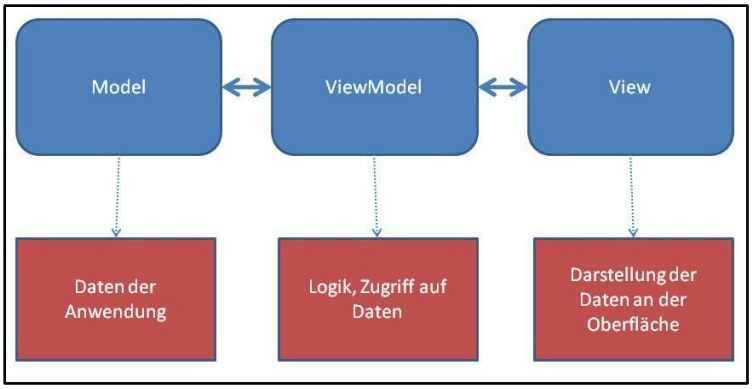


Abbildung 1: MVVM

Anschließend wurde für jedes dieser Modelle ein ViewModel erstellt, das die Daten der Modelle verfügbar macht. Auch diese ViewModelle werden direkt in C# entwickelt und müssen die Schnittstelle „*INotifyPropertyChange*“ implementieren, damit Änderungen an den Daten vom View gespeichert oder Änderungen am Modell im View sichtbar werden. Auf der Darstellungsschicht des Frameworks wird beispielsweise ein Benutzereingabefeld in XAML definiert, welches mittels Datenbindung an ein ViewModell gebunden wird. Nun wird der aktuelle Wert des Modells auf der Benutzeroberfläche sichtbar und Änderungen an dem Wert im Modell gespeichert. Genau dieses Verhalten ist für das Szenario des Reviews nötig. Ebenfalls der Framework-Charakter wird hierdurch verstärkt, da als die Darstellung gänzlich von der Struktur der Daten getrennt ist. Es ist möglich, die Modelle durch andere UML 2.0 Modelle zu ersetzten. Ebenfalls ist es möglich, die Darstellung der Daten beliebig zu variieren, indem für den jeweiligen Einsatzkontext spezialisierte Views erstellt werden (grafische Oberfläche) die dann mittels DataBinding auf die Daten des Modells zugreift.

Konkret implementiert wurde ein UML Klassendiagramm, da hier nur eine begrenzte Anzahl an Elementen verfügbar sind und somit die Komplexität nicht all zu hoch ist. Folgende Modelle wurden in C# für ein Klassendiagramm in den ersten Prototypen im Framework erstellt:

- UMLClass: Eine Klasse hat einen Namen und jeweils eine Liste von Attributen und Methoden
- UMLClassProperty: Dies stellt ein Attribut einer Klasse dar; Attribute werden derzeit dadurch gekennzeichnet, dass sie einen Typen und einen Bezeichner haben

- UMLClassOperation: Dies stellt eine Methode einer Klasse dar; Methoden werden derzeit dadurch gekennzeichnet, dass sie einen Rückgabewert, einen Bezeichner und eine Signatur haben
- UMLClassAssociation: Eine Beziehung zwischen zwei Klassen wird derzeit durch eine Assoziation dargestellt, diese hat zwei Kardinalitäten und einen Bezeichner
- UMLClassGeneralization: Eine Vererbungshierarchie wird hierdurch gekennzeichnet, es sind keine Werte hinterlegt, da dies lediglich über die grafische Repräsentation gekennzeichnet wird

Für jede dieser Modellklasse wurde anschließend ein ViewModel in C# erstellt, das jeweils das „*INotifyPropertyChange*“ Interface implementiert. Für jedes Feld innerhalb des ViewModels, hier nun beispielsweise für den Namen einer Klasse dargestellt, muss in etwa folgendes Schema angewendet werden:

```
private string _Name;
      public string Name
      {
          get { return _Name; }
          set
          {
              _Name = value;
              _Source.Name = value;
              RaisePropertyChanged("Name");
          }
      }
```

Der Name ist derzeit als String-Objekt hinterlegt. Sobald der Setter für dieses Feld aktiviert wird, wird der neue Wert im ViewModel als Name hinterlegt und `raisePropertyChanged("Name")`

ausgeführt, was bewirkt, dass das Model (in diesem Fall UMLClass) bzw. der entsprechende Wert aktualisiert wird. In diesem Fall handelt es sich um einen einzigen Wert, da jede UML Klasse nur einen einzigen Namen halten kann. Attribute und Methoden werden jedoch als Liste von Attributen und Methoden angelegt, da eine Klasse 0 – n Attribute/Methoden enthalten kann. Hierfür muss die Liste dem Typen „*ObservableCollection<ViewModel>*“ entsprechen, welches wiederum die ViewModel der Attribute/Methoden aktzeptiert. Die „*ObservableCollection*“ implementiert dasselbe Verhalten, wie das „*RaisePropertyChanged*“, indem die Änderungswerte an das entsprechende Modell weitergereicht wird.

Der View wird in XAML definiert. Hier wird sowohl die grafische Repräsentation in einer XML/HTMLähnlichen Notation als auch Datensetter und Datentrigger definiert. XAML-Elemente sind beliebig zu verschachteln und können je nach Situation im Programmfluss getauscht werden. Für die UML Klasse wurden beispielweise für jeden möglichen View (siehe 4.2) ein anderes Template erstellt. Die Verbindung zwischen View und ViewModel wird durch das Binden der Datan (DataBinding) gelöst.

4.2 Darstellung von Informationen auf begrenztem Raum und deren Handling

Bei der Erstellung von UML-Klassendiagrammen auf dem Microsoft Surface fiel schnell auf, dass eine einzelne Klasse recht viel Platz auf dem Surface in Anspruch nimmt. Es war auch nicht möglich, die Darstellung einfach kleiner zu gestalten, da die Interaktion mit dem Surface mittels Fingern geschieht und bei einer zu kleinen Darstellung die Genauigkeit nicht ausreicht. An einem Desktop-PC ist eine höhere Genauigkeit durch den Einsatz einer Maus zu erzielen, ein Finger ist jedoch nicht „so spitz zulaufend“ und somit würden evtl. mehrere aktivierbare Komponenten mit einem Finger berührt, was zu nicht gewünschtem Benutzungsverhalten führen würde.

Eine UML2.0-Klasse aus einem Klassendiagramm hält Informationen über Attribute und Methoden, sowie einen Namen. Diese Informationen sollen editierbar sein. Des Weiteren soll eine einzelne Klasse in Beziehungen zu anderen Klassen stehen (Verbindungen) und alle Klassen zusammen ergeben ein Gesamtbild. Wären all diese Informationen auf einmal sichtbar, wären Knöpfe für die Bearbeitung, Darstellung aller Attribute zugleich etc. nötig. Diese ganzen Informationen und Knöpfe müssen lesbar und eindeutig fokusierbar (Platz zwischen den Elementen nötig) sein. Unter diesen Voraussetzungen wären bis zu vier verschiedene Klassen auf dem Surface darstellbar. Da diese für den Entwurf eines Systems oder ähnlichem nicht ausreicht, musste herausgefunden werden, wie die Informationen geschickter verteilt werden können.

Eine Lösung, die zusammen mit dem Team erstellt wurde, ist, dass die immer nur ein Teil der gewünschten Informationen pro Element angezeigt wird. Dieses Vorgehen nennen wir „View“ eines Elements für den Benutzter. So soll der Benutzer zwischen den darzustellenden Informationen wählen können, die er derzeit benötigt. Für die Darstellung einer Klasse nutzen wir die Metapher einer Karte mit verschiedenen Seiten. Durch eine Aktion des Benutzers lässt sich die darzustellende Information auswählen.

Folgende Informationsdarstellungen wurden implementiert:

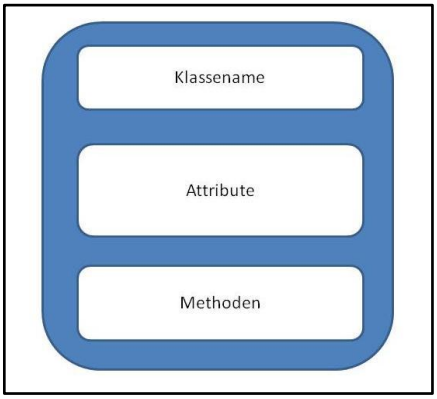


Abbildung 2: InformationsView

Informationsansicht: Die Attribute und Methoden als auch der Name der Klasse sind sichtbar. Attribute und Methoden sind als Liste angeordnet. Die Attribute und Methoden sind als Liste angezeigt, welche mittels Interaktion mit dem Finger selektiert werden kann.

Abbildung 3: ÜbersichtsView

Übersicht: Nur der Name der Klasse ist sichtbar, diese Ansicht eignet sich vor allem für die Strukturierung im Zusammenhang mit anderen Klassen.

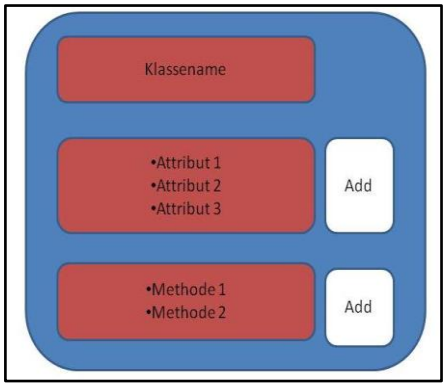


Abbildung 4: BearbeitungsView

Bearbeitung: In dieser Ansicht sind dieselben Informationen wie in der Informationsansicht verfügbar, jedoch anders farblich dargestellt und mit anderen möglichen Aktionen. So reagiert ein Attribut bei Kontakt mit dem Finger und die Bildschirmtastatur erscheint, nun kann ein bestehendes Attribut geändert werden. Ein Button am Rand ermöglicht es, neue Attribute oder Methoden zu der jeweiligen Liste hinzuzufügen. Diese sind genau wie beschrieben änderbar.

Bei der Arbeit mit den Views hat es sich herausgestellt, dass die Aktivierung von Events auf dem Microsoft Surface derzeit lediglich durch zwei Gesten gesteuert werden kann:

- Tab-Geste: Antippen eines Bereichs aktiviert einen Event
- Hold-Geste: Das Halten des Finger auf einer Position wird nach ca. 2-3 Sek. durch einen Event abgefangen

Die Hold-Geste dauert für einen flüssigen Ablauf zu lange, dies hat das Testen mit verschiedenen Personen gezeigt. Daher bleibt einzig die Tab-Geste übrig. Weitere Gesten werden nicht vom Microsoft Surface SDK unterstützt. Ein eigenes Gesten-Framework mit frei definierbaren Gesten ist nicht Bestandteil dieses Projekts, daher sollten die bereits verfügbaren Mittel genutzt werden. Ursprünglich war eine Idee, dass ein Element auf dem Surface überall „angefasst“ werden kann, um die Position zu ändern und eine weitere Geste zwischen den View umschaltet. Dieses Vorgehen war leider nicht zu realisieren, da die Elemente dazu größtenteils deaktiviert werden mussten und dann die Tab-Geste nicht mehr verfügbar war. Die Gesten hatten Einfluss auf die Entscheidungen zum Design der Views, daher werden diese an dieser Stelle hervorgehoben.

Folgende Schaubilder sollen dies verdeutlichen:

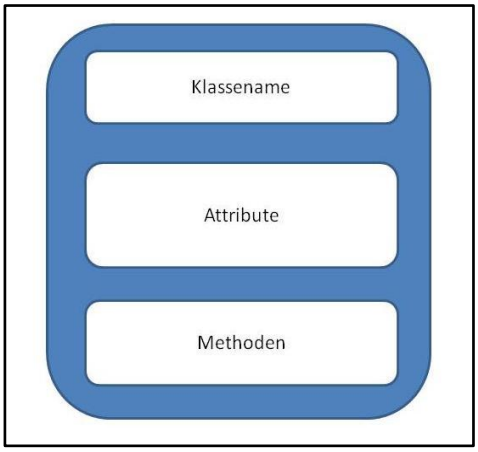


Abbildung 5: InformationsView Positionierung

Hier ist nochmals der InformationsView dargestellt und zeigt den stilisierten Aufbau einer Klasse auf der Benutzeroberfläche. Dabei sind die weißen Kästen mit Informationen gefüllt. Um die Position der Klasse zu ändern, legt der Benutzer seinen Finger auf eine beliebige Position der Klasse und verschiebt diese (*Drag and Drop*-Verhalten). Dazu müssen jedoch die Felder für Name, Attribute und Methoden auch auf das *Drag and Drop*-Verhalten reagieren. Dabei reagieren diese Felder nicht mehr auf die Tab-Geste, da das Surface nicht unterscheiden kann, ob der Benutzer nun verschieben oder bearbeiten möchte. Damit diese Entscheidung möglich ist, musste der Bereich, ähnlich wie bei Fenstern in gängigen Betriebssystemen, der zum Positionieren benötigt wird, eingeschränkt werden.

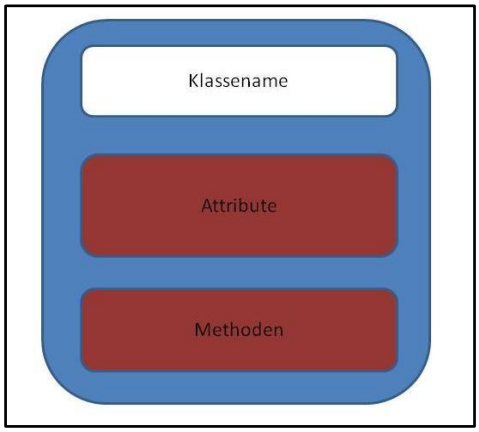


Abbildung 6: aktivierbare Bereiche

Abbildung 6 zeigt diese Aufteilung, Die rot markierten Bereiche reagieren nicht auf *Drag und Drop*, sondern auf die Tab-Geste. Sobald ein Benutzer mit seinem Finger diese Bereiche berührt, ändert sich die Ansicht der Klasse (in diesem Fall von Information- auf Bearbeitungsansicht). Der blaue und weiße Bereich wurde so konfiguriert, dass hier die Klasse nicht durch die Ansichten schaltet, sonder vom Benutzer verschoben werden kann. Dies wird visuell dadurch unterstützt, dass die Klasse dann mittels Schattendarstellung erhoben dargestellt wird.

Um viele Informationen auf einem begrenzten Raum wie dem Microsoft Surface, darzustellen, wurden die Informationen in sog. Views aufgeteilt. Zwischen diesen Views kann auf bestimmten, definierten Schaltflächen gewechselt werden. Die restliche Fläche kann durch den Benutzer zur Positionierung des gesamten Elements genutzt werden. Es werden für das gesamte Framework derzeit nur 2 Gesten unterstützt. Dabei liegt der Schwerpunt hier auf der Tab-Geste.

4.3 Interaktion mit realen Objekte auf dem Microsoft Surface

Das Microsoft Surface beinhaltet eine Technologie zur Erkennung von sog. Tags. Diese Tags stellen kleine Aufkleber dar, die eine eindeutige Nummer enthalten. Solche Aufkleber können an beliebigen realen Objekten angebracht werden, sodass das Surface diese erkennt. Für das Framework ist dies eine Möglichkeit, die Interaktion der Benutzer weiter zu vereinfachen als auch den benötigten Platz für viele Werkzeuge zu sparen.

Folgende Werkzeuge wurden im Framework durch reale Objekte und Tags realisiert:

- Ein Mülleimer für das Löschen von Elementen
- Eine Toolbox mit verschiedenen Werkzeugen

Das Benutzungsmodell im Umgang mit realen Objekten auf dem Surface soll durch folgendes Szenario beschrieben werden:

Am Rand der Oberfläche des Surface steht eine Figur mit der Form eines Mülleimers. Am unteren Ende ist diese Figur mit einem Aufkleber versehen. Der Benutzer hat eine Klasse vorliegen, die nicht mehr benötigt wird, um diese zu löschen, platziert er die Figur des Mülleimers auf der Monitorfläche. Das Surface erkennt anhand des Tags die Figur als Mülleimer und lässt ein Element erscheinen, in das Elemente gezogen werden können. Der Benutzer positioniert nun die zu löschende Klasse über dem Mülleimerelement (*Drag and Drop*). Sobald ein Element über dem Mülleimer losgelassen wird, wird dieses Element vom Surface entfernt und ist als Miniaturdarstellung im Mülleimer enthalten. Entfernt der Benutzer die Figur des Mülleimers wieder von der Oberfläche, wird durch einen *Fade-Out* Effekt ebenfalls die Darstellung des Mülleimers vom Monitor entfernt. Die gelöschte Klasse befindet sich nun nicht mehr im sichtbaren Bereich. Sollte die gelöschte Klasse doch noch verwendet werden, so muss die Figur des Mülleimers erneut auf dem Monitor platziert werden. Nun kann die

Miniaturansicht der gelöschten Klasse aus dem Mülleimerelement auf den Monitor „gezogen“ werden. Anschließend ist dieses Element wieder wie gewohnt zu behandeln.

Progammtechnisch handelt es sich bei diesem Verhalten um sog. *TagVisualizer*. Diese Klasse muss mit einem Wert versehen werden, auf die das Microsoft Surface reagiert und ebenfalls auf dem Aufkleber hinterlegt ist. Sobald der entsprechende Tag vom Surface registriert wird, erscheint das hinterlegte Template (in diesem Fall die Benutzungsschnittstelle eines Mülleimers) und sobald das Element vom Tisch genommen wird, wird auch der Software-Mülleimer vom Monitor entfernt. Verschiebt man die Position der Figur, wird auch die Position des Software-Mülleimers entsprechend geändert, ebenso wie bei Rotation dieses.

Der Vorteil für Applikationen auf dem Microsoft Surface ist, dass Standardfunktionen, wie Löschen, Hinzufügen etc., die nicht ständig genutzt werden, nur dann sichtbar ist, wenn der Benutzer dies wünscht. Es müssen auch nicht spezielle Gesten implementiert werden, sondern der Benutzer kann mit gewohnten Metaphern und realen Gegenständen die Applikation steuern, was aus Erfahrungen mit Nutzer recht intuitiv umgesetzt und schnell angenommen wird, indem er reale Gegenstände zur Interaktion nutzen kann.

Die Inhalte und Zustände der Softwareeinheiten der Figuren bleiben auch bestehen, wenn diese gerade nicht sichtbar bzw. die Figur nicht auf dem Monitor steht.

Um den Gedanken eines Frameworks beizubehalten und möglichst viele Varianten an Diagramme zu ermöglichen, wurden nur die Standardtoolbox und der Mülleimer als zwei Funktionen direkt in das Framework implementiert. Die Standardtoolbox umfasst derzeit folgende vier Buttons, die mittels *Expression Blend* erstellt wurden:

- Hinzufügen: Eine neues Default-Element wird dem Diagramm hinzugefügt
- Speichern: Das gesamte Diagramm wird mittels XMI gespeichert
- Ausrichten: Ein Button mit der Darstellung eines Magneten soll das gesamte Diagramm entsprechend ausrichten, diese Funktionalität ist noch nicht implementiert
- Öffnen: Ein in XMI gespeichertes Diagramm soll geladen werden

Weitergehende diagrammspezifische Funktionen müssen eigens für jeden Diagrammtyp erstellt werden. Dazu müssen zwei Komponenten erstellt werden:

- *TagVisualizationControl*
- *TagVisualizationDefinition*

Das *TagVisualizationControl* stellt die grafische Repräsentation, also den View dar. Hier wird definiert, wie das Element grafisch aussieht und welche Funktionen enthalten sind. Hier wird ähnliche wie unter 4.1. beschrieben ein Template in XAML erstellt, welche mittels Binding an ein ViewModel gebunden wird.

Die *TagVisualizationDefinition* stellt die Schnittstelle zum Microsoft Surface dar. Hier wird definiert, welcher Tag mit welchem Wert verwendet wird und einige weitere Einstellungen, wie z.B. das *Fade-Out und Fade-In* Verhalten. Des Weiteren können hier spezifische Events abgefangen werden. Im

aktuellen Prototyp müssen diese Elemente noch manuell und direkt in das Framework implementiert werden. Zukünftig soll es hierfür eine geeignete Schnittstelle geben.

5. Fazit

Das letzte Kapitel dieser Ausarbeitung zeigt zusammenfassend die Ergebnisse als auch Projekterfahrungen der ersten Hälfte des Projektunterrichts des Sommersemesters 2009 auf. Des Weiteren soll ein Ausblick über die zweite Hälfte dargestellt werden.

5.1 Zusammenfassung

Die Projektziele für einen multitouch-fähigen Prototypen einer Applikation, der im Kontext des Softwareerstellungszyklus unterstützen kann, und anhand dessen der Umgang der Entwicklung von Applikationen und die Möglichkeiten des Microsoft Surface evaluiert wurden, wurden umgesetzt. Der Umgang mit den Technologien und deren Besonderheiten, wie beispielsweise das DataBinding und das Erstellen von XAML-Templates für die Darstellung wurde im Team erarbeitet. Ebenso wurde anhand von Demos und eigenen Mini-Applikationen „probiert“, wie spezielle Hindernisse oder Probleme im Umgang mit dem Surface gelöst werden können. Das Resultat ist ein (Fast-)Framework, das im ersten Schritt UML Klassendiagramme darstellen kann, anhand dessen ganze Klassendiagramme erstellt oder bearbeitet werden können. Derzeit befindet sich das Framework noch in einem Status als Prototyp, es benötigt noch einige Refactorings um den Framework-Charakter heraus zu bilden. Dies wird jedoch Teil der zweiten Projekthälfte sein.

5.2 Projekterfahrung

An dieser Stelle möchte ich meine ganz persönlichen Erfahrungen dieses Projektes zusammenfassen. Es war sehr spannend, an einer neuen und noch nicht weit verbreiteten Technologie, wie dem Microsoft Surface, zu arbeiten. Alte Probleme, wie das *MVC-Pattern* oder *Drag and Drop*, musste neu gelöst werden, da der Umgang mit einem mutlitouch-fähigem Tisch neue Anforderungen an eine Applikation und dessen Umgang stellt. Ebenso spannend war es, dass die Probleme noch nicht weit verbreitet sind, da die Technologie noch sehr neu ist und somit die gelernten Mechanismen, wie z.B. Design-Pattern, angewandt werden mussten, um selbstständig eine Lösung der Probleme zu finden. Des Weiteren wurde das prototypische Entwickeln in kurzen Iterationen praktiziert, was sehr zu dem „Erfolg“ des Projekt beigetragen hat, da hier die Ideen und Lösungen mit allen Projektmitgliedern geteilt wurde und somit das erhaltene Wissen verteilt wurde.

Bei dem Umgang mit multitouch-fähigen Tischen fiel vor allem auf, dass gänzlich anderen Anforderungen an eine Applikation von den Benutzern gestellt werden, als an eine Desktop-Variante. Dies liegt zum einen daran, dass Eingaben mit einem oder mehr Finger gemacht werden, und somit keine Abstraktion der Eingabe, wie bei der Maus vorherrscht. Dies hat zur Folge, dass ein Benutzer die Elemente auf der Oberfläche intuitiv so nutzt, wie die Metapher des Elements auch in der realen Welt genutzt werden würde. Hier ist der Einsatz der Interaktion von realen Objekten und der Software besonders hilfreich. Massive Texteingaben als auch die Monitorfläche sind begrenzende Faktoren im Umgang mit dem Surface. Text lässt sich zwar über die Bildschirmtastatur eingeben, aber die dazu vom Benutzer eingenommene Haltung und die fehlende Haptik der Tastatur sind nicht optimal für die Eingabe von viel Text. Da keine Maus zur Interaktion mit dargestellten Elementen

eingesetzt wird, sonder die Finger der Benutzer, müssen die schaltbaren Elemente entsprechend groß sein, was eine Herausforderung an das Design einer Applikation stellt.

5.2.1 Bewerbung bei verschiedenen Konferenzen

Neben dem eigentlichen Projekt hat sich das gesamte Projektteam (Hauke Wittern, Daniel Löffelholz, Prof. Dr. Olaf Zukunft und Thorben Pergande) mit den Ergebnissen der Arbeit bei verschiedenen Konferenzen im Umfeld von Multitouch als Teilnehmer mit einem Konferenz-Papier beworben. Die Erfahrung der Erstellung solch eines Papiers, vor allem die geballten Informationen und Erfahrungen auf den Punkt in einen bestimmten Kontext zu stellen, war sehr fordernd als auch spannend zugleich.

5.3 Ausblick/ Erweiterungsmöglichkeiten

Im kommenden Semester (Wintersemester 2009/2010) wird dieses Projekt im Rahmen des Masterstudiums Informatik an der Hochschule für Angewandte Wissenschaften Hamburg fortgesetzt. Es steht noch aus, ob das Framework von Grund auf neu strukturiert (Framework-Charakter) oder mit den gesammelten Erfahrungen in den Kontext der Logistik gewechselt wird. Sollte das Framework im Fokus der zweiten Projekthälfte stehen, ist es wichtig, dass weitere Diagrammtypen unterstützt werden.