Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Projektbericht

Hauke Wittern

Entwicklung eines Frameworks für die Softwaremodellierung auf Multitouch Tischen

Fakultät Technik und Informatik
Department Informatik

18. September 2009

Inhaltsverzeichnis

1. Einleitung

Dieser Projektbericht fasst die Ergebnisse des Projekts zusammen, das der Autor zusammen mit Daniel Löffelholz und Thorben Pergande im Sommersemester 2009 an der Hochschule für Angewandte Wissenschaften Hamburg durchgeführt hat. Das Projekt wurde von Prof. Dr. Olaf Zukunft betreut. In diesem Projektbericht stehen die vom Autor behandelten Themen im Vordergrund.

Das Ergebnis des Projekts ist der Prototyp eines Frameworks für die Softwaremodellierung auf dem Microsoft Surface [URL:MSSurface] sowie die Referenzimplementierung für auf dem Framework aufbauenden Anwendungen. Das Framework hat den Namen COSMOS (COllaborative Surface for MOdelling Software). Auf dem Framework aufbauende Anwendungen werden im Folgenden als COSMOS Anwendungen bezeichnet. Die Referenzimplementierung für COSMOS Anwendungen ist eine Anwendung, mit der Softwareentwickler UML-Modelle erstellen und betrachten können. COSMOS verwendet Microsoft Surface Multitouch Tische als Zielplattform. Die Konzepte des Frameworks können aber auch auf andere Multitouch Tische übertragen werden, sofern die Merkmale der Hardware ähnlich sind.

Dieser Bericht erläutert zunächst in Kapitel 2 die Grundlagen, die für das Verständnis in den weiteren Kapiteln nötig sind. Kapitel 3 erläutert kurz die Ziele, die Vision und die Anforderungen des Projekts. Anschließend fasst der Hauptteil die Themen zusammen, mit denen sich der Autor während des Projekts auseinander gesetzt hat. Kapitel 4 beschreibt die Architektur des Frameworks und Kapitel 5 erläutert die wichtigsten Konzepte, die das Framework umsetzt. Danach gibt Kapitel 6 einen Überblick über den Projektablauf. Kapitel 7 schließt diesen Bericht mit einer Zusammenfassung des Projekts und einem Ausblick auf mögliche Weiterentwicklungen ab.

2. Grundlagen

Dieses Kapitel erläutert die Grundlagen, die zum Verständnis dieses Berichts nötig sind. Der erste Abschnitt stellt das Microsoft Surface vor. Anschließend werden Domänenspezifische Sprachen und Reviews definiert.

2.1. Der Microsoft Surface Multitouch Tisch

Das Microsoft Surface [URL:MSSurface] ist ein tischförmiger Computer dessen Tischplatte ein Multitouch fähiges Display ist. Mehrere Benutzer können das Microsoft Surface gleichzeitig durch Berührung des Displays mit mehreren Händen bzw. Fingern und Gegenständen bedienen. Die einzelnen Berührungspunkte werden als Kontakte bezeichnet. Die Erkennung der Kontakte erfolgt mit Hilfe von Infrarotkameras, die sich unter dem Display befinden. Herkömmliche Eingabegeräte wie Maus und Tastatur sind für die Bedienung vom Hersteller Microsoft nicht vorgesehen [Microsoft Corporation (2009)].

Die visuelle Erkennung erlaubt es, die Art der Kontakte näher zu bestimmen. Das Microsoft Surface SDK unterstützt die Klassifizierung von Kontakten als Finger, physikalisches Objekt oder getagtes physikalisches Objekt. Getagte Objekt sind Gegenstände, die mit einem Aufkleber versehen sind. Ein solcher Aufkleber kodiert entweder den Objekttyp (Byte-Tag) oder die Identität eines bestimmten Objekts (Identity-Tag). Die Idee Gegenstände zur Interaktion zu verwenden stammt von Fitzmaurice et.al., welche solche Gegenstände als «graspable objects» (greifbare Objekte) bezeichnen [Fitzmaurice u. a. (1995)].

2.2. Domänenspezifische Sprachen (DSL)

Eine Domänenspezifische Sprachen (engl. domain-specific language, DSL) ist eine formale Sprache zur Beschreibung von Modellen aus einer bestimmten Domäne. Eine Domänenspezifische Sprachen setzt sich aus einem Metamodell und einer konkreten Syntax zusammen. [Stahl u. a. (2007)]

Das Metamodell ist eine formale Beschreibung einer Domäne und besteht aus einer abstrakten Syntax und einer statische Semantik. Die abstrakte Syntax definiert die Elemente der Domäne und welche Beziehungen zwischen den Elementen bestehen können. Welche Bedingungen die Elemente und Beziehungen einhalten müssen, definiert die statische Semantik. [Stahl u. a. (2007)]

Die konkrete Syntax einer DSL legt fest, wie die Elemente der abstrakten Syntax aussehen. Eine konkrete Syntax kann entweder grafisch oder textuell sein. Bei der Modellierung mit einer Domänenspezifische Sprache werden die Modelle mit der konkreten Syntax erstellt. [Stahl u. a. (2007)]

2.3. Reviews

Im Bereich der Softwareentwicklung versteht man unter einem Review eine Methode, mit der mehrere Personen ein Dokument einer manuellen Prüfung unterziehen. Reviews nutzen die Menschliche Analyse- und Denkfähigkeit, um komplexe Sachverhalte zu prüfen und zu bewerten. Häufig sind sie die einzige Möglichkeit, die Semantik von Dokumenten zu überprüfen. Der Vorteil von Reviews ist frühzeitig Fehler aufdecken zu können. Damit können sie die Kosten zur Fehlererkennung und Beseitigung vermeiden. [International Software Testing Qualifications Board (ISTQB) (2007), Spillner und Linz (2005)]

Alle Dokumente die während des Softwareentwicklungsprozesses erstellt werden, können mit Reviews überprüft werden. Die Teilnehmer eines Reviews können sowohl Entwickler als auch Auftraggeber sein. So kann z.B. Quelltext von Entwicklern auf semantische Fehler überprüft werden. Außerdem können Entwickler zusammen mit den Auftraggebern Modelle des geplanten Systems oder Anforderungsspezifikationen auf Unstimmigkeiten untersuchen.

3. Projekt Ziele

Dieses Kapitel definiert die Ziele des Projekts und nennt die wichtigsten dabei auftretenden Anforderungen. Zuerst wird die Vision der Modellerierung von Software mit dem Microsoft Surface dargelegt und Reviews als ein konkretes Einsatzszenario aufgezeigt. Anschließend werden die Anforderungen, unterteilt in fachliche und technische Anforderungen, aufgelistet.

3.1. Vision

Das Ziel des Projekts ist die Entwicklung von Konzepten für Werkzeuge, die Softwareentwickler bei der Modellierung von Software auf Multitouch Tischen verwenden können. Das Ergebnis des Projekts soll ein Prototyp eines Frameworks sein, welches als Basis für, auf dem Microsoft Surface verwendbaren, Modellierungswerkzeuge dienen kann. Ein Prototyp eines Werkzeugs für Reviews von UML-Klassendiagrammen soll den Einsatz des Frameworks demonstrieren.

Multitouch Tische unterscheiden sich grundlegend von klassischen Desktop Computern und Notebooks. Sie ermöglichen neue Szenarien des computerunterstützten Arbeitens, einige Tätigkeiten können sie aber nicht so gut unterstützen wie klassische Computer. Da Multitouch Tische eine neue Technologie sind und auf dem Markt kaum verfügbar sind, ist ihr Einsatz bisher noch nicht ausreichend erforscht. Die am Projekt beteiligten Personen interessieren sich sehr für das Softwareengineering und haben sich deshalb Entschieden zu untersuchen, wie und wann Softwareentwickler Multitouch Tische bei der Modellierung von Software einsetzen können.

Die wesentlichen Unterschiede zwischen Multitouch Tischen und klassischen Computern sind bei den Eingabemethoden und den Arbeitsumgebungen zu finden. Touchfähige Geräte können die Benutzer mit ihren Fingern und physikalischen Objekten bedienen, statt mit Maus und Tastatur. Bei multitouchfähigen Geräten können auch mehrere Benutzer gleichzeitig Eingaben tätigen. Die Eingabe mit den Fingern ist allerdings weniger präzise als mit der Maus. Die Eingabe von Text ohne ein Tastatur ist ein weiteres Problem. Klassische Computer sind aufgrund der Eingabemethoden und ihrer Bauform am besten für Ein-Personen-Arbeitsplätze geeignet sind. Multitouch Tische ermöglichen es hingegen mehreren Personen, gemeinsam in einer Runde zusammenzuarbeiten.

Die Projektmitglieder sehen in der Möglichkeit des gemeinsamen Arbeitens den größten Vorteil von Multitouch Tischen. Das Framework soll deshalb besonders die kollaborativen Tätigkeiten während des Softwareentwicklungsprozessen unterstützen.

3.2. Szenario

Das konkrete Szenario, mit dem ein Prototyp den Einsatz des Frameworks demonstrieren soll, sind Reviews von UML-Modellen. Bei einem solchen Review besprechen die Teilnehmer ein zu-

vor erstelltes Modell, um es zu verstehen oder, um unpassende Entwürfe zu finden und alternativen zu finden.

Reviews wurden als Szenario gewählt, da sie zum einen in einer Sitzung mit mehreren Teilnehmern durchgeführt werden, während der die Teilnehmer kollaborativ Arbeiten. Zum anderen sehen die Projektmitglieder ein großes Potential dafür, die Produktivität von Reviews durch den Einsatz von Multitouch Tischen zu steigern.

Multitouch Tische können eine lockere und produktive Arbeitsatmosphere fördern, da die Teilnehmer bequem in einer Runde um den Tisch sitzen können. Das Sitzen in einer solchen Runde ist bei der Verwendung von herkömmlichen Computern, Projektoren oder Whiteboards nicht möglich, wenn die Teilnehmer gemeinsam das untersuchte Dokument betrachten sollen. Diese Geräte erforden es, dass die Teilnehmer entweder sehr nah beieinander sitzen müssen oder in eine Richtung, ausgerichtet auf eine Wand und den Vortragenden, blicken müssen. Dies kann zu einer für die Teilnehmer unbehaglichen Atmosphere führen. Die Atmosphäre während eines Reviews ist ohnehin schon kritisch, denn der Autor des Dokuments könnte in eine psychologische Stresssituation gelangen, sollte er sich persönlich angegriffen fühlen [Spillner und Linz (2005)]. Die Sitzordnung um einen Tisch kann solche Probleme vermeiden. Der Einsatz von Multitouch Tischen verspricht außerdem persönliche Angriffe zu vermeiden, da die Benutzer stark auf die dargestellten Inhalte fokussiert sind. Diese Fokussierung auf den Inhalt haben die Projektmitglieder bei allen Surface Anwendungen beobachtet. Der Grund dafür liegt in der Attraktivität einer Surface Anwendung, welche die Benutzer zur Interaktion einlädt.

3.3. 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-Modelierungs-Werkzeug. Im Folgenden werden zunächst die fachlichen und anschließend die technischen Anforderungen vorgestellt.

3.3.1. 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 (siehe 3.2).
- Die Benutzer müssen alle Funktionen der Anwendung mit den Fingern und mit auf dem Surface befindlichen physikalischen Gegenständen bedienen können. Herkömmliche Eingabegeräte, 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 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.

3.3.2. Technische Anforderungen

Die in diesem Projekt verwendete Hardware ist ein Microsoft Surface [URL:MSSurface] 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 (siehe [Microsoft Corporation (2009)]) 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 .Net Frameworks.
- Als Implementierungssprachen sollen C# und XAML (Extensible Application Markup Language) verwendet werden.

4. Architektur des COSMOS Frameworks

Die Architektur des COSMOS-Frameworks ist komponentenorientiert und schichtenartig aufgebaut. Die Wiederverwendbarkeit der Komponenten minimiert den Aufwand bei der Entwicklung neuer Anwendungen. Bei Bedarf können Anwendungsentwickler die im Framework vorhandenen Komponenten anpassen oder gegen eigene Implementierungen austauschen.

Dieses Kapitel gibt einen Überblick über die Architektur des COSMOS-Frameworks. Zunächst wird erläutert, wie das Framework aufgebaut ist und anschließend wird am Beispiel eines UML-Editors gezeigt, wie eine auf dem Framework basierende Anwendung aufgebaut ist.

4.1. Schichten bei Verwendung des COSMOS Frameworks

Die Komponenten einer mit COSMOS entwickelten Anwendung können vier verschiedenen Schichten zugeordnet werden (siehe Abb. 1):

1. Core Layer,
2. Common Layer,
3. Domain Layer,
4. Application Layer

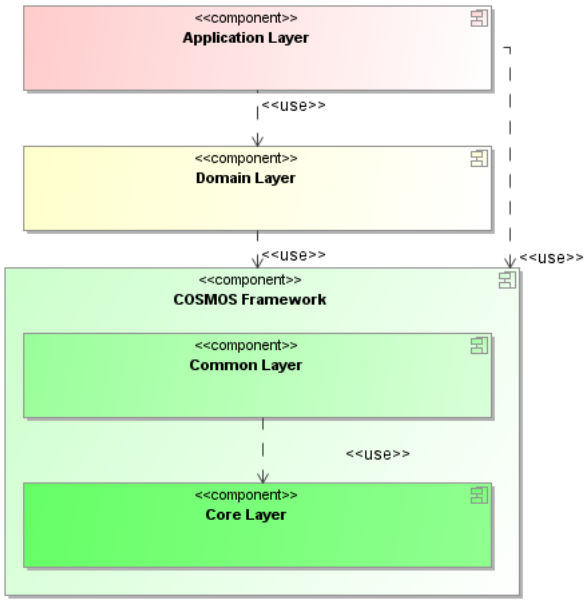

Abbildung 1: Die vier Schichten einer COSMOS-Anwendung

Diese Schichten kapseln die Konzepte des Frameworks auf verschiedenen Abstraktionsstufen. Die Entwickler können das Framework auf der höchstmöglichen Abstraktionsstuffe verwenden, ohne alle Details des Frameworks verstehen zu müssen. Gleichzeitig erlaubt es das Framework den Entwicklern, bei Bedarf zusätzliche Funktionalität auf allen Abstraktionsstuffen hinzuzufügen und per Dependency Injection Standardimplementierungen auszutauschen.

Das Framework selbst besteht aus zwei Schichten: dem Kern des Frameworks (Core Layer) und allgemeinen Standardkomponenten (Common Layer). Das Core Layer implementiert die Kernkonzepte des Frameworks und bildet damit das Fundament für alle weiteren Komponenten. Zu den Kernkonzepten gehören die Beschreibung von Domänen, die Abbildung von Modellen auf Diagramme und das Werkzeugmodell. Auf diese Konzepte geht Kapitel 5 später im Detail ein. Die höheren Schichten verwenden die vom Core Layer bereitgestellten Konzepte, um die Funktionalität von Anwendungen zu implementieren. Das Common Layer enthält Komponenten mit der Funktionalität, welche die meisten Anwendungen benötigen. Dazu gehören beispielsweise Werkzeuge wie der Papierkorb und die standardmäßigen visuellen Effekte der Benutzungschnittstelle.

Auf dem Framework bauen die Implementierungen von Domänenspezifischen Sprachen (DSL) als dritte Schicht auf (Domain Layer). Zurzeit sind UML-Klassendiagramme als Referenzimplementierung einer solchen DSL vorhanden. Die UML-Komponente und jede andere DSL-Komponente sind wiederverwendbar. Die Komponenten des Domain Layers zählen aber nicht zum Framework, da sie keine neue grundlegende Funktionalität bereitstellen.

Die oberste Schicht bildet schließlich die konkrete Anwendung (Application Layer). In dieser Schicht implementieren die Anwendungsentwickler die Anwendungslogik eines Editors und teilen dem Framework die verwendeten Komponenten per Dependency Injection mit.

4.2. Komponenten eines Editors

Die Konfiguration jedes mit COSMOS entwickelten Editors muss angeben, welche Domänen unterstützt werden und wie diese angezeigt bzw. editiert werden können. Ein COSMOS-Editor besteht aus folgenden Komponenten:

- Eine Definition der abstrakten Syntax der Domäne.
- Eine Implementierung der statische Semantic.
- Eine zur abstrakten Syntax passende grafische konkrete Syntax. Diese beschreibt, wie die Elemente der Domäne in der Benutzungsoberfläche dargestellt werden.
- Eine Sammlung von Werkzeugen, die kollaboratives Arbeiten mit den Diagrammen unterstützen. Die Werkzeuge können sowohl domänenspezifisch als auch domänenunspezifisch sein können.
- Die Anwendungslogik.
- Die Anwendungskonfiguration.

Die abstrakte Syntax und die statische Semantik repräsentieren das Metamodell einer Domäne. Zusammen mit der konkreten Syntax bilden sie eine vom Editor verwendbare domänenspezifische Sprache.

Die Entwickler einer neuen Anwendung müssen zumindest die Anwendungskonfiguration und die Anwendungslogik implementieren. Wenn die Anwendung eine DSL und Werkzeuge wiederverwendet, ist die Entwicklung weiterer Komponenten nicht erforderlich.

Abbildung 2 zeigt die technische Architektur des UML Editors, welcher die Referenzimplementierung für COSMOS Anwendungen ist.

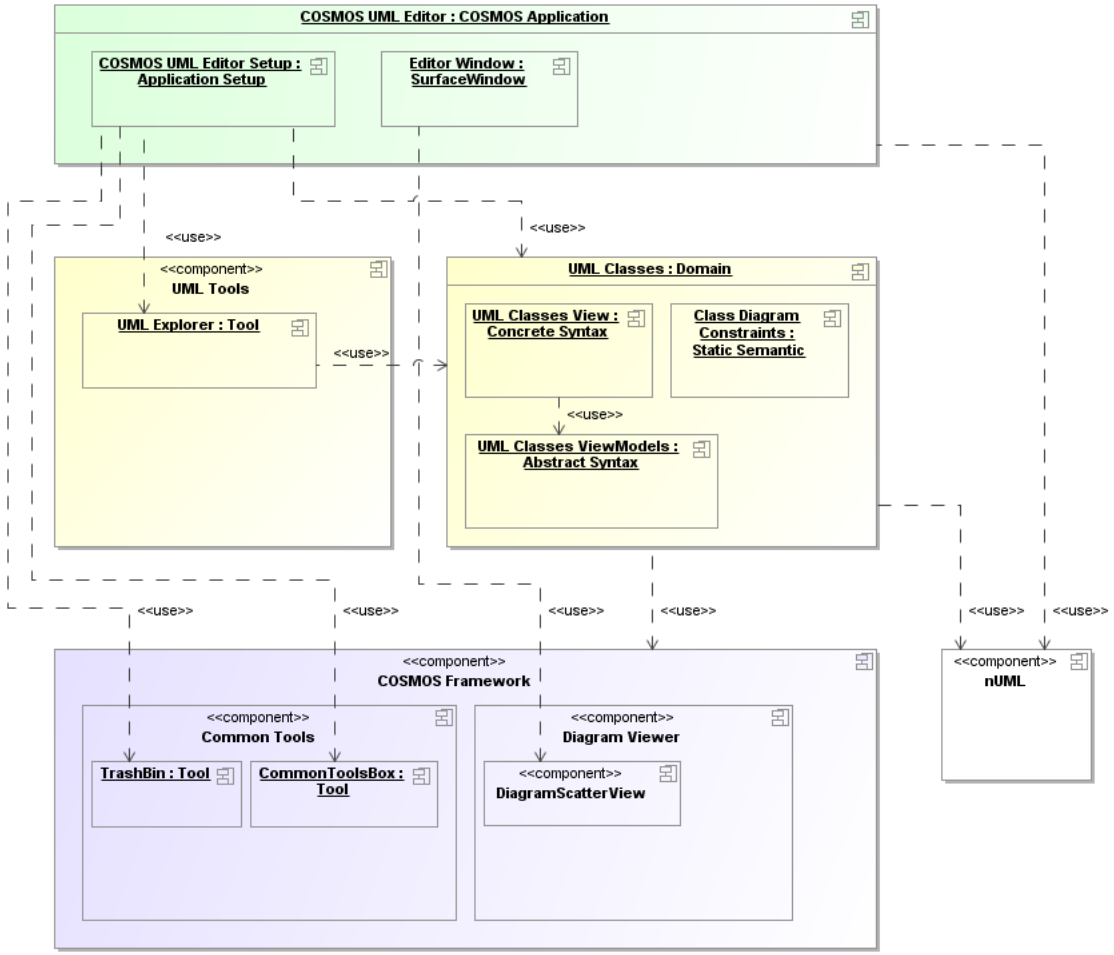

Abbildung 2: Technische Architektur des UML Editors

5. Konzepte des COSMOS Frameworks

Dieses Kapitel gibt einen Überblick über die wichtigsten vom COSMOS-Framework bereitgestellten Konzepte. Als erstes wird die Konfiguration von Anwendungen erläutert. Danach wie Do-

mänenspäzifische Sprachen und Werkzeuge implementiert werden. Zum Schluss werden die in COSMOS enthaltenen Abstraktionsmechanismen und das Konzept der visuellen Gedankenführung vorgestellt.

5.1. Konfiguration von COSMOS-Anwendungen

Die zentrale Komponente einer COSMOS-Anwendung ist die Anwendungskonfiguration. Die Laufzeitumgebung des Frameworks ermittelt anhand der Anwendungskonfiguration, welche Domänen unterstützt werden und wie die zugehörigen Domänenspezifischen Sprachen aussehen sowie welche Werkzeuge verfügbar sind.

Ein Entwickler implementiert die Anwendungskonfiguration mit der Extensible Application Markup Language (XAML) (siehe Listing 2 im Anhang). XAML ist eine XML basierte Sprache, die es aufgrund ihres deklarativen Stils erlaubt, die Anwendungskonfiguration auf einfache Weise und leicht verständlich zu implementieren. XAML erlaubt es Ressourcen zu deklarieren und wiederzuverwenden. Dies reduziert sowohl den Entwicklungsaufwand als auch die Größe der einzelnen Konfigurationsdateien. Letzteres erhöht die Lesbarkeit der XAML Dateien. COSMOS verwendet solche Ressourcen, um unter anderem Domänenspezifische Sprachen und Werkzeuge wiederzuverwenden.

5.2. Abbildung von Modellen auf Diagramme

Das COSMOS Framework unterstützt Diagramme in Form von Graphen mit Knoten und Kanten. Ein Modell, das als Diagram dargestellt werden soll, muss deshalb auf einen Graph abgebildet werden. Wie diese Abbildung aussieht kann ein Domänenentwickler deklarativ mit XAML festlegen. Die Deklaration der Abbildung besteht aus einer Liste von Knoten Definitionen (NodeDefinition) und Kanten Definitionen (EdgeDefinition), denen der Typ des abzubildenden Domänenelements zugewiesen wird (siehe Listing 1 im Anhang).

5.3. Definition einer domänenspezifischen Sprache

Die Implementierung einer von COSMOS unterstützten domänenspezifischen Sprache (DSL) orientiert sich am MVVM Pattern (Model-View-ViewModel). Die abstrakte Syntax einer Domäne ist dabei das Model und die konkrete Syntax ist die View. Das ViewModel ist das von der View verwendete Model und wird als ein Adapter [Gamma u. a. (2004)] für die abstrakte Syntax implementiert.

5.3.1. Definition der abstrakten Syntax

Als abstrakte Syntax kann bei COSMOS eine beliebige Implementierung einer Domäne wiederverwendet werden. Diese übernimmt die Rolle des Models. Dieses Model ist jedoch nicht dafür geeignet, von der View verwendet zu werden. Deshalb wird ein ViewModel als Adapter verwendet.

COSMOS verwendet WPF als Technologie für die Benutzungsoberfläche. Dabei kommen Konzepte wie das Data-Binding zum Einsatz, die vom Model in der Regel nicht unterstützt werden. Ein ViewModel macht als Adapter diese Konzepte für das Model verfügbar.

Außerdem kann die Struktur der abstrakten Syntax nicht mit der in Abschnitt 5.2 beschriebenen Abbildung von Modellen auf Diagramme zusammenpassen. Es kann Domänenelemente geben, die nicht anhand ihres Typs eindeutig auf einen Knotentyp bzw. einen Kantentyp abgebildet werden können. Beispielsweise gibt es bei UML Klassendiagrammen verschiedene Arten von Assoziationen (Aggregation, Komposition etc.), die alle durch Domänenelemente vom Typ Assoziation repräsentiert und anhand ihrer Attribute (AggregationKind) unterschieden werden. Jedoch sollen Aggregationen und Kompositionen im Graphen unterschiedlich dargestellt werden.

Mit dem Wissen über die Domäne kann ein Entwickler einer Domänenspezifischen Sprache verschiedene Adapter für ein Domänenelement entwickeln. Diese Adapter ermöglichen eine korrekte Abbildung eines Domänenelements auf einen Knotentyp bzw. einen Kantentyp. Der Typ des Adapters muss dann in der Knoten bzw. Kanten Definition angegeben werden.

5.3.2. Definition der statischen Semantik

Ein auf COSMOS basierender Editor muss sicherzustellen, dass die dargestellten Diagramme formal korrekt sind. Beispielsweise dürfen bei Petri Netzen Stellen mit Transitionen verbunden werden nicht aber mit Stellen. Der Editor erlaubt es dem Benutzer nicht solche fehlerhaften Modelle zu erstellen. Welche Manipulationen ein formal korrektes Modell bewahren, ermittelt der Editor anhand der statischen Semantik, welche Bedingungen gegen die abstrakte Syntax definiert.

Das COSMOS Framework stellt Schnittstellen für die statische Semantik bereit, die ein Domänenentwicklern implementieren muss. Gegebenenfalls können die Schnittstellen als Adapter implementiert werden, um eine vorhandene Implementierung der statischen Semantik wiederzuverwenden. Die statische Semantik wird in der mit XAML geschriebenen Domänen-Definition angegeben.

5.3.3. Definition der konkreten Syntax

Die konkrete Syntax entspricht der View und legt fest, wie die Elemente einer abstrakten Syntax tatsächlich aussehen. Die konkrete Syntax wird bei COSMOS mit den DataTemplate und Style Konzepten der WPF implementiert.

Mit DataTemplates kann ein Domänenentwickler das Aussehen von Domänenelementen, die das Framework auf Knoten abbildet, beschreiben. Ein DataTemplate wird mit den Controls der WPF implementiert. Das Aussehen der Verbindungslinien und Endpunkte von Kanten legt der Domänenentwickler mit Styles fest. Die DataTemplates und Styles werden in der Domänendefinition den Knotendefinitionen bzw. den Kantendefinitionen zugewiesen.

5.4. Implementierung von Werkzeugen

Werkzeuge, die kollabratives Arbeiten unterstützen, sind ein zentraler Bestandteil des COSMOS Frameworks. Ein Werkzeug ist immer einem physikalischen Gegenstand zugeordnet. Mit Hilfe des Microsoft Surface SDKs kann die Anwendung diese Gegenstände anhand eines Aufklebers (Tag) erkennen. Wenn ein Benutzer die Funktionen eines Werkzeugs benutzen möchte stellt er den Gegenstand, der dem Werkzeug zugeordnet ist, auf das Surface. Daraufhin zeigt die Benutzungsoberfläche der Anwendung, dort wo der Gegenstand platziert wurde, ein virtuelles Werkzeug an. Um möglichst viel Bildschirmfläche zum anzeigen der Diagramme nutzen zu können, kann der Benutzer den Gegenstand und damit das virtuelle Werkzeug wieder entfernt werden. Da der Gegenstand weiterhin für alle Benutzer sichtbar ist, bleibt den Benutzern nicht verborgen, welche Werkzeuge es gibt.

Das COSMOS Framework beinhaltet einige Standardwerkzeuge, wie einen Mülleimer und ein Werkzeug zum hinzufügen neuer Diagrammelemente. Darüberhinaus kann ein Entwickler eigene Werkzeuge entwickeln. Dazu muss die Implementierung der Anwendungslogik des Werkzeugs von der abstrakten Framework-Klasse «Tool»erben. Zusätzlich muss der Werkzeug-Entwickler eine grafische Benutzungsoberfläche für die Anwendungslogik des Tools implementieren. Dies ist eine Spezialisierung der TagVisualization Klasse aus dem Surface SDK. Um das Tool in einer Anwendung verfügbar zu machen, muss der Entwickler in der Anwendungskonfiguration eine neue ToolDefinition hinzufügen und mit dieser sowohl die Tool-Klasse als auch die TagVisualization, inklusive des zugeordneten Tags, angeben.

5.5. Abstraktionsmechanismen und visuelle Gedankenführung

Ein Schlüssel für das einfache Verständnis komplexer Modelle sind Abstraktionsmechanismen und die visuelle Gedankenführung. Das COSMOS Framework ermöglicht es deshalb dem Benutzer, zwischen verschiedenen Abstraktionsstufen zu navigieren und unwichtige Details auszublenden. Visuelle Effekte sollen es darüberhinaus den Betrachtern der Modelle erleichtern, Änderungen der Sicht auf die Modelle zu verfolgen und zu verstehen.

COSMOS unterstützt verschiedene Abstraktionsstufen durch das Verschachteln von Diagrammen. Dies erlaubt es durch hierarchische Kompositionen zu navigieren. Ein Diagramm höheren Abstraktionsgrades stellt seine eingebetteten weniger abstrakten Diagramme als Knoten dar. Ein solcher Knoten kann von einem Benutzer als das Anzuzeigende Diagramm ausgewählt werden (implementiert durch eine Double-Tap Geeste). Die Navigation zu einer höheren Abstraktionsstufe kann mit einem Werkzeug durchgeführt werden. In zukünftigen Versionen von COSMOS ist hierfür alternativ eine Geeste vorgesehen.

Ein weiterer Abstraktionsmechanismus sind die verschiedenen Sichten (Views) einzelner Diagrammelemente mit unterschiedlichen Detailgraden. Mit dem Ansicht-Werkzeug von COSMOS können die Benutzer die angezeigten Details auch für alle Diagrammelemente eingestellen.

Alle anderen Änderungen der Anzeige soll die Benutzungsoberfläche animiert darstellen. Die Animationen verhindern das abrupte Erscheinen bzw. Verschwinden von Elementen der Benutzungsoberfläche und fördern damit das Verständnis der Betrachter [Microsoft Corporation (2009)].

Die Navigation zwischen den Diagrammen illustriert die Benutzungsoberfläche durch eine animierte Explosion eines Diagramm-Knotens in seine einzelnen Elemente bzw. durch verschmelzen der Elemente eines Diagrammes in einen Knoten. Diese Metaphern helfen den Betrachtern den Wechsel des angezeigten Diagramms nachzuvollziehen und sind ähnlich wie die Zemantic Zooming Technik von Frisch et al. [Frisch und Dachselt (2008), Frisch u. a. (2008)]. Den Wechsel der Sicht illustriert die Benutzungsoberfläche mit der Metapher des Umdrehens einer Spielkarte.

6. Projektablauf

Rückblickend betrachtet kann der Ablauf des Projekts in vier Phasen unterteilt werden: die Einarbeitungsphase, die Implementierungsphase, das Schreiben eines Papers und die Refaktorisierung und Weiterentwicklung.

Am Anfang des Projekts war die Microsoft Surface Technologie für alle Beteiligten neu. Die Projektmitglieder benötigten deshalb einige Zeit, um die Richtung festzulegen, in die das Projekt laufen sollte. Bei der Auslotung der Möglichkeiten des Surface half die Entwicklung eines rudimentären Prototyps einer Anwendung zum Austauschen von Kontaktinformationen. Hierbei hat weniger die technische Umsetzung sondern vielmehr das Gespräch der Projektmitglieder untereinander geholfen. Die Diskussionen der Projektmitglieder haben schnell ergeben, dass sie ein großes Potential für hochgradig kollaboratives Arbeiten an Multitouch Tischen sehen. Da sich die Projektmitglieder sehr für das Software Engineering interessieren, haben sie daraufhin beschlossen, eine UML-Modellierungsanwendung für das Surface zu entwickeln.

Die Anfangsphase des Projekts war außerdem stark durch den Wissensaustausch geprägt. Bis auf den Autor hatten die Projektmitglieder kaum Erfahrung in C# und mit der WPF sowie XAML. Der Autor hatte bereits langjährige Erfahrung mit diesen Sprachen und Technologien. Diesen

Kontrast beim Wissensstand haben die Projektmitglieder am Anfang des Projekts durch Pair- und Tripple-Programming versucht zu reduzieren. Im weiteren Verlauf haben sie den Wissensaustausch untereinander und auch mit einer anderen Gruppe von Studenten vorgesetzt. Dadurch ist es gelungen den Wissensstand zumindest so anzugleichen, das alle Projektmitglieder selbstständig mit den Technologien Arbeiten können.

Mit dem steigenden Verständnis der Surface Konzepte und Technolgien ging die erste Einarbeitungsphase fließend in die Implementierungsphase über, in der ein großer Teil der Konzepte von COSMOS entwickelt wurde. Bis in die Implementierungsphase hinein hatten die Projektmitglieder allerdings damit zu kämpfen, kein funktionierendes Versionierungssystem zur Verfügung zu haben. Aufgrund von technischen Problemen bei der Installation des Team Foundation Servers, konnte erst nach 7 Wochen der Quellkode automatisch versioniert werden. Bis dahin litt die Produktivität sehr stark darunter, alle Änderungen von Hand integrieren zu müssen und zu verteilen. Bei der Installation des Team Foundation Servers hat der Autor die Erfahrung gemacht, dass man vor der Installation solcher komplexen und an Experten gerichteten Software die Dokumentation sorgfältig lesen sollte und jeden Setup-Schritt sorgfältig durchführen sollte. Das unüberlegte Übernehmen der Standardeinstellungen in den Setup-Wizards kann bei solcher Software zu einer nicht mit der gegebenen IT-Infrastruktur kompatiblen Konfigurationen führen.

Die größte Produktivität haben die Projektmitglieder erreicht, als sie beschlossen ein Paper zu Schreiben und bei Konferenzen einzureichen. Die Aussicht, an einer Konferenz teilzunehmen und die nahende Abgabefrist haben die Projektmitglieder sehr motiviert. In dieser Phase wurden in kurzer Zeit noch viele Ideen implementiert.

Nachdem das Paper abgegeben wurde, haben die Projektmitglieder noch nicht fertige Funktionalität implementiert und Refaktorisierungen durchgeführt. Die Refaktorisierungen waren nötig geworden, weil insbesondere während des Paper Schreibens einige Funktionen, die unbedingt demonstriert werden sollte, nicht sauber, teilweise hartkodiert und mit vielen Abhängigkeiten implementiert wurde. Jedoch haben diese unsauberen explorativen Prototypen geholfen Ideen auszuprobieren und zu zeigen, dass sie funktionieren.

7. Zusammenfassung und Ausblick

In diesem Projekt wurde ein Framework für die Modellierung von Software auf dem Microsoft Surface entwickelt und mit einer UML-Modellierungsanwendung demonstriert. Dieser Bericht hat die Architektur und die Konzepte des Frameworks hervorgehoben. Die Unterstützung von kollaborativer Arbeit und die Förderung der Kommunikation zwischen den Benutzern sehen die Projektmitglieder als gewichtige Vorteile von Modellierungsanwendungen auf Multitouch Tischen. Diese Vorteile versprechen die Produktivität der Benutzer zu erhöhen. Die Projektmitglieder sehen besonders bei Review-Szenarien ein Potential für Produktivitätssteigerungen.

Das Ergebnis des Projekts kann als Basis für Weiterentwicklungen im Projekt des nächsten Semesters dienen. Einen zwingenden Beweis für die versprochenen Vorteile, bei der Softwaremo-

dellierung auf Multitouch Tischen, ist das Projekt bis jetzt allerdings schuldig geblieben. Bei der Fortführung des Projekts hat deshalb Evaluierung der Beispielanwendung eine besonders hohe Priorität.

Mögliche Themen bei der Weiterentwicklung des Frameworks sind zusätzliche Abstraktionsmechanismen und die Generierung von Prototypen aus Modellen und deren Ausführung auf einem Multitouch Tisch. Außerdem könnte der Einsatz von Geesten genauer untersucht werden. Das Framework verwendet bisher nur wenige Standardgeesten.

A. Listings

Listing 1: Konfiguration einer Domäne mit XAML

```
<domDef:Domain
    DomainAdapter="{StaticResource {x:Type local:UMLClassesAdapter}}">

   <!-- Definiere Knoten und Kanten -->
   <domDef:Domain.ElementDefinitions>
      <domDef:NodeDefinition
            MappedType="{x:Type uml:Class}"
            Template="{StaticResource ClassRootTemplate}" />
      <domDef:EdgeDefinition
            MappedType="{x:Type uml:Generalization}"
            SourceArrowStyle="{StaticResource Arrow}"
            LineStyle="{StaticResource ConnectionLineStyle}">
         <domDef:EdgeDefinition.ConnectionSyntax>
            <cosmos:SimpleEdgeConnectionSyntax
                    AllowedSourceTypes="{StaticResource {x:Static
                          local:DomainDefinition.GeneralizableElementsKey}}"
                    AllowedTargetTypes="{StaticResource {x:Static
                          local:DomainDefinition.SpecializableElementsKey}}" />
         </domDef:EdgeDefinition.ConnectionSyntax>
      </domDef:EdgeDefinition>

      <!-- ... -->

   </domDef:Domain.ElementDefinitions>
   <!-- ... -->
</domDef:Domain>
```

Listing 2: Konfiguration einer Anwendung mit XAML

```
<cw:Application>
    <!-- ... -->
    <cw:Application.Setup>
        <appDef:ApplicationSetup
            ViewsConfiguration="{StaticResource {x:Type views:ViewsConfiguration}}">

            <!-- Die Verfügbaren Domänen dieser Anwendung: -->
            <appDef:ApplicationSetup.SupportedDomains>
                <StaticResource ResourceKey="{x:Static
                     uml:DomainDefinition.DefaultClassDiagramDefinitionKey}" />
            </appDef:ApplicationSetup.SupportedDomains>

            <!-- Die Tools -->
            <appDef:ApplicationSetup.ToolDefinitions>

                <!-- Mülleimer -->
                <StaticResource ResourceKey="DiagramElementTrashBinDefinition" />

                <!-- ToolBox mit Standard Tools -->
                <StaticResource ResourceKey="CommonToolBoxDefinition" />

                <!-- Ein Tool zum anpassen der Ansicht -->
                <StaticResource ResourceKey="DiagramViewSelectorToolDefinition" />

            </appDef:ApplicationSetup.ToolDefinitions>
        </appDef:ApplicationSetup>
    </cw:Application.Setup>
</cw:Application>
```

Listing 3: Konfiguration von Tools mit XAML

```
<ResourceDictionary>
    <!-- ***  Deklaration von im ApplicationSetup verwendeten Resourcen  *** -->

    <!-- Default Konfiguration des Mülleimer Tags -->
    <appDef:ToolDefinition x:Key="DiagramElementTrashBinDefinition"
                           ToolLogic="{StaticResource {x:Type ct:DiagramElementTrashBin}}">
        <appDef:ToolDefinition.Visualization>
            <s:ByteTagVisualizationDefinition
                 Source="DiagramElementTrashBinTagVisualization.xaml"
                 Value="0xC2" />
        </appDef:ToolDefinition.Visualization>
    </appDef:ToolDefinition>

    <!-- Default Konfiguration des CommonToolBox Tags -->
    <appDef:ToolDefinition x:Key="CommonToolBoxDefinition"
                           ToolLogic="{StaticResource CommonToolBox}">
        <appDef:ToolDefinition.Visualization>
            <s:ByteTagVisualizationDefinition Source="CommonToolBoxTagVisualization.xaml"
                                              Value="0xC1" />
        </appDef:ToolDefinition.Visualization>
    </appDef:ToolDefinition>

    <!-- Default Konfiguration des View Tags -->
    <appDef:ToolDefinition x:Key="DiagramViewSelectorToolDefinition">
        <appDef:ToolDefinition.Visualization>
            <s:ByteTagVisualizationDefinition
                  Source="DiagramViewSelectorTagVisualization.xaml" Value="0xC3" />
        </appDef:ToolDefinition.Visualization>
    </appDef:ToolDefinition>

  <!-- *** Instanziierung der Tool-Anwendungslogik, Templates, ToolDefinitions etc. *** -->

    <!-- Instanz der Anwendungslogik des Mülleimers -->
    <ct:DiagramElementTrashBin x:Key="{x:Type ct:DiagramElementTrashBin}" />

    <!-- *** Tools in der CommonToolbox *** -->
    <t:NewItemTool x:Key="{x:Type t:NewItemTool}" />
    <t:HistoryTool x:Key="{x:Type t:HistoryTool}" />

    <!-- Templates für Tools in der CommonToolbox -->
    <DataTemplate x:Key="AddNewItemToolViewTemplate">
        <tc:AddNewItemToolControl />
    </DataTemplate>

    <DataTemplate x:Key="HistoryToolViewTemplate">
        <tc:HistoryToolControl />
    </DataTemplate>

    <!-- Deklaration der Tools für die CommonToolbox -->
    <t:CommonToolBoxToolDefinition x:Key="AddNewItemToolDefinition"
             ViewTemplate="{StaticResource AddNewItemToolViewTemplate}"
             ToolLogic="{StaticResource {x:Type t:NewItemTool}}" Header="new item palette" />

    <t:CommonToolBoxToolDefinition x:Key="HistoryToolDefinition"
             ViewTemplate="{StaticResource HistoryToolViewTemplate}"
             ToolLogic="{StaticResource {x:Type t:HistoryTool}}" Header="History" />

    <!-- Instanz der Anwendungslogik der CommonToolBox -->
    <t:CommonToolBox x:Key="CommonToolBox">
        <t:CommonToolBox.Tools>
            <StaticResource ResourceKey="AddNewItemToolDefinition" />
            <StaticResource ResourceKey="HistoryToolDefinition" />
        </t:CommonToolBox.Tools>
    </t:CommonToolBox>
</ResourceDictionary>
```

Literatur

[Fitzmaurice u. a. 1995] FITZMAURICE, George W. ; ISHII, Hiroshi ; BUXTON, William A. S.: Bricks: laying the foundations for graspable user interfaces. In: *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM Press/Addison-Wesley Publishing Co., 1995, S. 442–449. – ISBN 0-201-84705-1

[Frisch und Dachselt 2008] FRISCH, Mathias ; DACHSELT, Raimund: Benefits of interactive display environments in the software development process. In: *CHASE '08: Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering*. New York, NY, USA : ACM, 2008, S. 53–56. – ISBN 978-1-60558-039-5

[Frisch u. a. 2008] FRISCH, Mathias ; DACHSELT, Raimund ; BRÜCKMANN, Tobias: Towards seamless semantic zooming techniques for UML diagrams. In: *SoftVis '08: Proceedings of the 4th ACM symposium on Software visualization*. New York, NY, USA : ACM, 2008, S. 207–208. – ISBN 978-1-60558-112-5

[Gamma u. a. 2004] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Entwurfsmuster*. Addison-Weslay Verlag, 2004. – ISBN 978-3-8273-2199-2

[International Software Testing Qualifications Board (ISTQB) 2007] INTERNATIONAL SOFTWARE TESTING QUALIFICATIONS BOARD (ISTQB): *Standard glossary of terms used in Software Testing*. 2007. – URL `http://www.istqb.org/downloads/glossary-current.pdf`

[Microsoft Corporation 2009] MICROSOFT CORPORATION: *Microsoft Surface User Experience Guidelines*. 06 2009. – URL `http://community.surface.com/downloads/p/156.aspx`

[Spillner und Linz 2005] SPILLNER, Andreas ; LINZ, Tiulo: *Basiswissen Softwaretest*. 3., überarbeitete und aktualisierte Auflage, korrigierter Nachdruck 2007. dpunkt.verlag GmbH, Heidelberg, 2005. – ISBN 3-89864-358-1

[Stahl u. a. 2007] STAHL, Thomas ; VÖLTER, Markus ; EFFTINGE, Sven ; HAASE, Arno: *Modellgetriebene Softwareentwicklung – Techniken, Engineering, Management*. 2. Auflage. dpunkt.verlag, Stuttgart, 2007. – ISBN 978-3-89864-448-8

[URL:MSSurface] MICROSOFT CORPORATION: *Microsoft Surface*. – URL `http://www.microsoft.com/surface`