Kompatibilitätsanalyse bei Evolution framework-basierter Anwendungen

Fabian Christ
s-lab - Software Quality Lab
Universität Paderborn
fchrist@s-lab.upb.de

Jan-Christopher Bals
Universität Paderborn
jan-christopher.bals@upb.de

Abstract: Die Entwicklung betrieblicher Informationssysteme basiert auf dem Einsatz von Frameworks. Diese bieten ein hohes Maß an Wiederverwendung und sind flexibel anpassbar. Mit der Evolution der eingesetzten Frameworks unabhängig von der Anwendung entsteht die Notwendigkeit, Frameworks durch neuere Versionen zu ersetzen, um Fehler zu beheben oder neue Funktionen benutzen zu können. Etwaige Inkompatibilitäten neuer Versionen erfordern Anpassungen an der Anwendung. In der Praxis entsteht das Problem, dass die erforderlichen Anpassungen schwierig zu bestimmen sind. In dieser Arbeit zeigen wir einen Ansatz zur automatischen Kompatibilitätsanalyse bei der Evolution framework-basierter Anwendungen.

1 Einführung

Ein Unternehmen, das die Entwicklung eines neuen betrieblichen Informationssystems in Auftrag gibt, erwartet, dass dies zu einem fest kalkulierten Preis und definierten Zieltermin geschehen kann. Damit dies für den Softwarehersteller möglich wird, muss dieser viele Teile des neuen Systems aus bereits existierenden Komponenten aufbauen, um bei der Entwicklung nur noch kundenspezifische Teile ergänzen zu müssen. Zentrales Konzept ist der Einsatz von Frameworks, die als Gerüste bereits Teile des Systems implementieren und nur noch durch eigene Erweiterungen spezialisiert werden müssen.

In der Literatur finden sich eine Reihe von Definitionen für Frameworks [JF88, Deu89, Sch97], wobei wir uns auf eine Definition für "Application Frameworks" von Johnson aus dem Jahre 1997 beziehen wollen [Joh97]. Danach ist ein Framework a) definiert durch seine wiederverwendbare Struktur: "A framework is a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact." und b) durch seinen Zweck als domänenspezifisches anpassbares Gerüst einer Anwendung: "A framework is the skeleton of an application that can be customized by an application developer.". Die Anpassung erfolgt dabei an so genannten Erweiterungspunkten, für die entsprechende Erweiterungen entwickelt werden können, die das Verhalten der Anwendung steuern. Aufgrund des hohen Grads an Wiederverwendungsmöglichkeiten sind Frameworks zentraler Bestandteil heutiger Softwaresysteme.

Wir betrachten in Abbildung 1 das Beispiel einer Anwendung, das ein Framework für die Programmierung der grafischen Benutzeroberfläche (GUI) verwendet. Das Framework

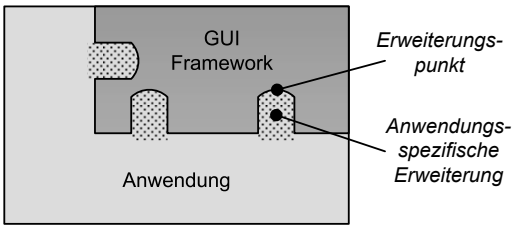

Abbildung 1: Anwendung verwendet ein GUI-Framework

bietet die Möglichkeit vorgefertigte Steuerelemente wie Dialoge und Listen zu verwenden. Mit Hilfe entsprechender *Erweiterungspunkte* des Frameworks können eigene neue Steuerelemente implementiert werden.

Nachdem eine Anwendung mit Hilfe des Frameworks entwickelt und ausgeliefert wurde, betritt sie die Weiterentwicklungsphase im Entwicklungszyklus. Jedoch nicht nur die Anwendung im Sinne anwendungsspezifischer Erweiterungen wird weiterentwickelt, sondern auch das eingesetzte Framework evolviert unabhängig davon. Unterschiedliche Entwicklungs- und Releasezyklen führen dazu, dass sich die beteiligten Teile unterschiedlich schnell weiterentwickeln. Es kommt zu der Situation, dass für geplante Funktionen der Anwendung das Framework durch eine neue Version aktualisiert werden muss. Jedoch stellt sich die Frage, welche Auswirkungen eine solche Migration der Anwendung hat.

In Abbildung 2 betrachten wir das Problem einer Anwendung in Version 2.3.0, die ein GUI-Framework in Version 1.1.5 verwendet. Nun soll die Anwendung zu Version 2.4.0 weiterentwickelt werden und dabei ggf. auch das GUI-Framework von Version 1.1.5 auf Version 1.2.0 aktualisiert werden. Ob während der Weiterentwicklung der Anwendung von Version 2.3.0 auf 2.4.0 das GUI-Framework aktualisiert wird, hängt davon ab, ob die neue Frameworkversion ebenfalls kompatibel zur bestehenden Anwendung ist. Je nachdem wie viel Aufwand es bedeuten würde, entstehende Inkompatibilitäten zu beheben, wird man sich unter Zeit- und Kostendruck für oder gegen eine Migration entscheiden.

Die grundlegende Frage im dargestellten Problem bezieht sich darauf, ob die bestehende Anwendung kompatibel zur neuen Frameworkversion ist. Wir wollen daher die ge-

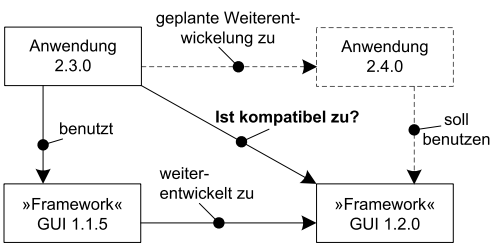

Abbildung 2: Problemstellung

plante Weiterentwicklung der Anwendung außer Acht lassen und das Problem darauf einschränken, dass die Anwendung als unverändert betrachtet wird und das Framework aktualisiert werden muss.

Um die Kompatibilität zu beurteilen, sucht man nach auswertbaren Informationen und stößt dabei meist schnell an Grenzen, da in der Regel der Fall einer speziellen Migration nicht dokumentiert ist. Häufig finden sich gar keine Informationen, die eine verlässliche Beurteilung der Situation ermöglichen würde. Aus diesen Gründen ist die industrielle Praxis in einer solchen Situation, dass man einen oder mehrere Entwickler abstellt, um die Migration probehalber durchzuführen. Dies bedeutet, dass die Entwickler in einem manuellen Prozess versuchen die Anwendung mit dem neuen Framework zu übersetzen und zu starten. Auftretende Fehlermeldungen werden manuell analysiert, im Internet recherchiert und mögliche Lösungen gesucht. Ziel ist es, wieder eine lauffähige Anwendung zu erhalten, so dass man anschließend eventuell vorhandene funktionale Tests durchführen kann. Dieses Vorgehen hat eine Reihe von Nachteilen:

- Es ist ein manueller und damit zeit- sowie kostenaufwendiger Prozess.
- Entwickler werden gebunden, die ansonsten für andere (wichtigere) Entwicklungen eingesetzt werden könnten.
- Der Aufwand einer Migration auf Probe ist im Worst-Case genauso hoch wie die einer tatsächlichen Migration. Es entsteht Aufwand, den man eigentlich erst investieren möchte, nachdem man sich endgültig für eine Migration entschieden hat.

Als Alternative zu diesem Vorgehen wünscht man sich ein System, das die Frage nach der Kompatibilität selbstständig entscheidet, ohne dass hierzu ein Entwickler benötigt wird. Da das selbstständige Entscheiden in der Praxis wohl kaum zu lösen sein wird, fokussieren wir uns in dieser Arbeit darauf, durch automatische Analyseverfahren mögliche Einschränkungen der Kompatibilität zu bestimmen, die einem Projektleiter als belastbare Grundlage dienen, die anstehende Migration zu planen und im Sinne einer Aufwandschätzung zu bewerten.

Im folgenden Abschnitt 2 werden wir unseren Ansatz[1], den wir im Rahmen der Entwicklung eines Software-Stacks für semantische Content-Management-Systeme entworfen haben, detailliert erläutern und verwandte Arbeiten betrachten. Unser Ansatz verwendet eine Kombination aus statischer Codeanalyse in Zusammenspiel mit der Auswertung von Frameworkdokumentationen basierend auf dem “Framework Description Meta-Model” (FDMM) [CBE+10]. Im Abschnitt 3 werden wir das FDMM kurz vorstellen, um dann in Abschnitt 4 auf die Kompatibilitätsanalyse einzugehen. Abschnitt 5 ist der Beschreibung unserer Implementierung vorbehalten. Wir schließen diese Arbeit mit einer kurzen Diskussion und einem Ausblick in Abschnitt 6.

[1]This work has been part-funded by the European Commission under grant agreement FP7-ICT-2007-3/ No. 231527 (IKS - Interactive Knowledge Stack).

2 Ansatz

Ein Hauptgrund für die gängige Praxis einer “Migration auf Probe” bei Frameworkaktualisierungen, ist die meist mangelhafte Dokumentation von Frameworks bzw., falls eine ausreichende Dokumentation vorliegt, das Fehlen einer Möglichkeit diese Informationen automatisch auswerten zu können. Aus diesem Grund haben wir in einem ersten Schritt ein Meta-Modell für Frameworkdokumentationen (FDMM) entwickelt, das in Abschnitt 3 genauer betrachtet wird. Die Motivation zur Erstellung eines neuen Meta-Modells war, dass die existierenden Ansätze in diesem Bereich zum einen nicht alle aus unserer Sicht erforderlichen Aspekte einer Frameworkdokumentation abdecken konnten und zum anderen, dass es keinen Ansatz gab, der Informationen so erfassen konnte, dass diese für Analyseprozesse automatisch verarbeitbar gewesen wären. Durch unsere Erfahrungen in der Meta-Modellierung entschlossen wir uns, die etablierten Ideen aus bestehenden Ansätzen mit erweiterten Möglichkeiten in ein vereinheitlichendes Modell zu überführen. Das FDMM definiert eine Sprache für Frameworkdokumentationen und macht Vorgaben an Struktur und Inhalt. Das Meta-Modell als formale Grundlage dieser Sprachdefinition gestattet es uns, eine FDMM-basierte Frameworkdokumentation automatisch verarbeiten zu können.

Da die Erstellung einer entsprechenden Dokumentation, obwohl in Forschung und Praxis als sehr wichtig angesehen [Par11], eine zu häufig unbeachtete Aufgabe ist, kann man hinterfragen, ob in der Praxis eine FDMM-basierte Dokumentation erstellt werden wird. Wir glauben jedoch, dass erst die Auswertbarkeit der Informationen für einen Anwendungszweck, wie den hier beschriebenen, den Wille zur Dokumentation steigern wird. Mit dem FDMM können wir zeigen wie Meta-Modell basierte Sprachen helfen, Informationen weiterverarbeiten zu können und so Mehrwerte zu erzeugen.

Unser Lösungsansatz in Abbildung 3 basiert darauf, dass beide Frameworkversionen mit Hilfe des FDMM dokumentiert wurden. Die Dokumentation wird verwendet, um durch Vergleich die Unterschiede zwischen zwei Frameworkversionen zu extrahieren. Die Differenz zwischen den Dokumentationen, gemeinsam mit Informationen über die Benutzung, ermöglicht uns eine neue Form der Bewertung der Kompatibilitätsfrage.

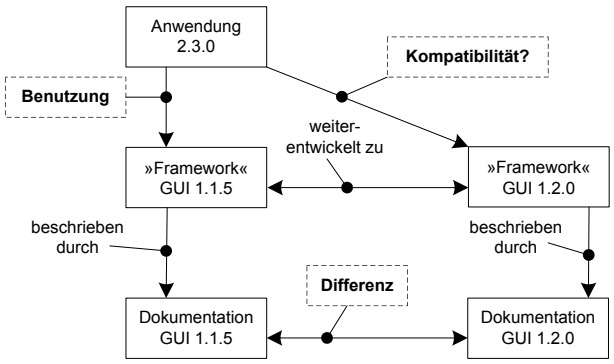

Abbildung 3: Lösungsansatz

Neben einer FDMM-basierten Dokumentation des Frameworks benötigen wir Informationen darüber, welche Teile des Frameworks von der Anwendung benutzt werden. Diese Benutzung könnte ebenfalls in auswertbarer Form dokumentiert worden sein. Wir gehen jedoch davon aus, dass eine solche Information nicht immer vorliegen wird, so dass wir statische Codeanalysen zur Analys der Bezüge zwischen Anwendung und Framework einsetzen. Diese Bezüge entstehen, weil z.B. eine Erweiterung in der Anwendung eine Schnittstelle implementiert, die von einem Framework definiert wird. Dies ist ein Hinweis darauf, dass die Anwendung an dieser Stelle einen Erweiterungspunkt des Frameworks benutzt. Neben der Analyse des Quellcodes können zusätzlich noch Ressourcen wie Konfigurationsdateien analysiert werden. Ziel dieser *Benutzungsanalyse* ist herauszufinden, welche Erweiterungspunkte des Frameworks von der Anwendung benutzt werden.

Parallel zur Benutzungsanalyse können wir die FDMM-basierte Dokumentationen beider Versionen in einer *Differenzanalyse* miteinander vergleichen und Unterschiede ermitteln. Anschließend können wir durch Kombination der Ergebnisse Rückschlüsse auf die Kompatibilität ziehen. Hierzu führen wir eine *Kompatibilitätsanalyse* in Bezug auf die verwendeten Erweiterungspunkte durch und bewerten gefundene Unterschiede im Hinblick auf zu erwartende Kompatibilitätsprobleme. Die Bewertung während der Kompatibilitätsanalyse liefert als Ergebnis eine Liste von Einschränkungen, die die Kompatibilität entweder potentiell oder in jedem Fall brechen. Potentiell brechende Einschränkungen sind jene, bei denen ein Bruch der Kompatibilität vorliegen könnte, wenn zum Beispiel das Framework durch neue Funktionen erweitert wird. Ein Beispiel für eine brechende Einschränkungen ist das Löschen eines bisher verwendeten Erweiterungspunktes.

Im Ergebnis dieser Analyseschritte, die in Abbildung 4 noch einmal dargestellt sind, liefert unser Ansatz eine Liste von Einschränkungen, die zum Beispiel von einem Projektleiter dazu verwendet werden kann, um die Auswirkungen einer Frameworkaktualisierung beurteilen zu können. Diese Auswirkungen können wiederum im Sinne von Aufwand bewertet werden, so dass man ohne Ausprobieren, eine Entscheidungsgrundlage hat, um für oder gegen eine Aktualisierung zu sein.

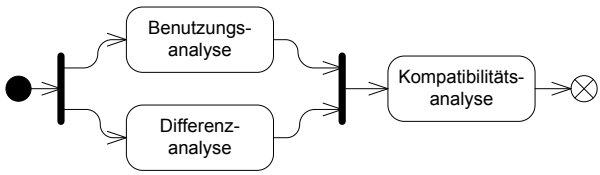

Abbildung 4: Analyseprozess

Das Thema Kompatibilität geht bereits auf Arbeiten aus dem Jahr 1968 [Gos68] zurück und hat viele Facetten. Verwandte Arbeiten für den Bereich der Kompatibilitätsanalyse beziehen sich meist auf das Problem der Kompatibilität von Komponenten und Services. Hier werden bereitgestellte und benötigte Schnittstellen betrachtet, um Aussagen über die Kompatibilität zu treffen, z.B. in [CZ06]. Eine Übersicht existierender Ansätze bietet [Cla06], wobei insbesondere die Arbeit von McCamant und Ernst [ME04] zu benennen ist, die Aussagen zur Kompatibilität bei Aktualisierung einer Komponente darüber ableiten, dass

sie das Verhalten beider Komponenten auf Basis von Tests protokollieren. Unser Ansatz geht davon aus, dass eine valide Dokumentation eine bessere Quelle für auswertbare Informationen darstellt, da in einer Dokumentation leichter alle Veränderungen erfasst werden können, als dass diese durch Tests und Beobachtung aufgedeckt werden könnten.

3 Meta-Modell zur Dokumentation von Frameworks

Für die Modellierung des FDMM zur Dokumentation von Frameworks haben wir nach eingehender Analyse existierender Frameworks und bestehender Ansätze zur Frameworkdokumentation eine Reihe von Anforderungen an Frameworkdokumentationssprachen ermittelt, die mit dem FDMM umgesetzt wurden. Ein Auszug der in [CBE+10] gestellten Anforderungen, die in Form von Merkmalen bezüglich einer relevanten Fragestellung aufgenommen wurden, zeigt Tabelle 1. Wir beschränken uns hier auf die Merkmale, die für eine Differenzanalyse im Hinblick auf die Kompatibilität von Interesse sind.

ID	Merkmal	Fragestellung
M1	Statische und dynamische Architekturübersicht	Wie ist die Gesamtarchitektur des Frameworks und an welchen Stellen befinden sich Erweiterungspunkte? Wie ist der Prozessfluss zwischen Framework und Erweiterungen?
M2	Erweiterungspunkt	Welche Erweiterungspunkte bietet das Framework an welchen Stellen in der Gesamtarchitektur und wie können diese genutzt werden?
M2.1	Bindung	Wie entsteht die Bindung zwischen einer Erweiterung und dem Framework, so dass die Erweiterung entsprechend eingebunden und verwendet wird?
M2.2	Installation	Wie und in welchem Format muss ein Erweiterung im Framework installiert (engl. deployed) werden?
M2.3	Protokoll	Welches Schnittstellenprotokoll muss eingehalten und bei der eigenen Implementierung einer Erweiterung berücksichtigt werden?

Tabelle 1: Dokumentationsmerkmal

Verwandte Ansätze im Bereich der Frameworkdokumentation arbeiten problemorientiert nach dem Kochbuchprinzip [SSP95, MCK97, OCM99], wobei ein häufig auftretendes Problem beschrieben und dann ein Rezept für dessen Lösung mit Hilfe des Frameworks vorgeschlagen wird. Ein Problem bei diesem Vorgehen ist, dass nicht jedes Problem in einer solchen Sammlung verfügbar ist und man bei Abweichungen vom Problem eine andere Form von Referenz vermisst. Diese Form von Unvollständigkeit zusammen mit dem Problem, dass es keine automatisch auswertbare Struktur in diesen Rezepten gibt, machen diese Ansätze unbrauchbar für eine Differenzanalyse, wie sie hier angestrebt wird.

Andere Ansätze entwickelten eigene Sprachen für die Dokumentation. Beispiele sind UML-basierte [OMG10] Sprachen wie die “UML-F” [FPR00, FPR02] – ein UML-Profil zur Dokumentation von Frameworkarchitekturen, die “UML-FD” [LSTM06] oder die “F-UML”

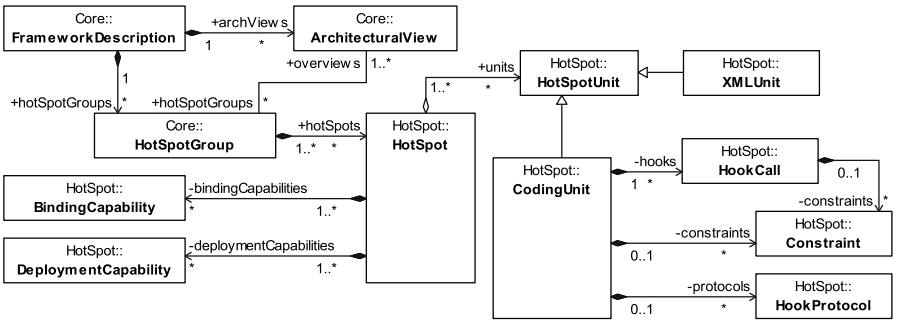

Abbildung 5: Auszug aus dem FDMM

[BBAGH03]. All diese Sprachen haben den Nachteil, dass sie nach den in Tabelle 1 aufgestellten Anforderungen unvollständig sind (siehe [CBE+10] für eine vergleichende Darstellung). Aus diesen Gründen haben wir das nachfolgend beschriebene FDMM entwickelt, das viele Ideen bestehender Ansätze in einem neuen Entwurf vereint und außerdem alle relevanten Merkmale einer Frameworkdokumentation abdeckt.

Das FDMM ist ein modular aufgebautes MOF-Modell [OMG06] dessen Kern aus drei Paketen besteht. Wir wollen hier aus Platzgründen nur die elementaren Teile des FDMM, dargestellt in Abbildung 5, kurz erläutern und den Bezug zu den Anforderungen aus Tabelle 1 herstellen. Im Paket `Basis` werden einige grundlegende Elemente definiert, wie eine Basisklasse für alle benannten Elemente. Das `Core` Paket enthält die Kernelemente von Frameworkdokumentationssprachen. Neben dem Wurzelelement `FrameworkDescription` gehören hierzu Sichten auf die Frameworkarchitektur, die über das Element `ArchitecturalView` (M1) repräsentiert werden. Erweiterungspunkte (`HotSpot`) werden gruppiert (`HotSpotGroup`) und mit der Architektur verlinkt.

Die Details eines `HotSpots` (M2) sind im gleichnamigen dritten Paket spezifiziert. Zu jedem `HotSpot` kann die Umsetzung des Bindens (`BindingCapability`) (M2.1) an das Framework und des Installierens (`DeploymentCapability`) (M2.2) im Framework dokumentiert werden. Ein `HotSpot` besteht aus einer Reihe von Untereinheiten (`HotSpotUnit`), die weitere Details zur Benutzung dokumentieren. Eine spezielle Form sind die `CodingUnits`, die zu implementierende Anteile beschreiben. Neben `CodingUnits` sind auch andere Formen denkbar, bei denen man zum Beispiel eine Erweiterung erstellt, indem man nicht im eigentlichen Sinne programmiert, sondern etwa eine XML-Datei mit entsprechendem Inhalt anlegt (`XMLUnit`). Eine `CodingUnit` beschreibt eine Reihe von zu implementierenden Methoden, genannt `HookCalls`, die vom Framework aufgerufen werden, um die Erweiterung zu bedienen. Solche `HookCalls` und die `CodingUnit` selbst können einer Reihe von Einschränkungen (`Constraints`) unterliegen, die zum Beispiel den Wertebereich der Parameter eines `HookCalls` einschränken. Eine weitere wichtige Eigenschaft ist das Kommunikationsprotokoll (`HookProtocol`) (M2.3) zwischen einem Erweiterungspunkt und einer Erweiterung beschreiben zu können. Es gilt zu beachten, dass hinter den hier gezeigten Elementen, wie einem `HookCall`

oder einem `HookProtocol`, sich weitere Elemente befinden. Um etwa die Details eines `HookProtocols` beschreiben zu können, benötigt man eine eigene Sprache wie UML-Protokollzustandsautomaten, die in das FDMM integriert werden kann. Für weitere Details verweisen wir auf [CBE+10]. Ein Dokumentationsbeispiel werden wir im nachfolgenden Abschnitt 4 geben.

4 Kompatibilitätsanalyse

Die Kompatibilitätsanalyse verwendet die Informationen aus Benutzungs- und Differenzanalyse. Die Benutzungsanalyse liefert Bezüge zwischen Anwendung und Framework, so dass die verwendeten Erweiterungspunkte identifiziert werden können. Die Differenzanalyse ermittelt alle Unterschiede zwischen zwei Versionen der Frameworkdokumentation. In der Kompatibilitätsanalyse wird geprüft, ob sich Änderungen bei verwendeten Erweiterungspunkten ergeben haben. Diese Änderungen werden im Hinblick auf die Kompatibilität bewertet und eine Liste von Einschränkungen der Kompatibilität generiert.

Wir werden dieses Vorgehen an einem vereinfachten Praxisbeispiel aus dem Bereich Web-Anwendungen verdeutlichen. Das Open-Source Framework "Apache MyFaces"[2] ist eine Implementierung der "Java Server Faces" (JSF) Spezifikation[3]. Mit diesem Framework lassen sich zum Beispiel GUI-Elemente wie Listen, Tabellen und Formulare leicht auf einer Web-Seite darstellen. Um eigene GUI-Elemente zu entwerfen bietet das Framework den Erweiterungspunkt `HS-UIComponent` an. Die Dokumentation in Abbildung 6 für MyFaces Version 1.1.5 besagt, dass man für diesen Erweiterungspunkt von der Klasse `UIComponent` erben. Mit der Vererbung muss man zusätzlich die Hook-Methoden `getValueBinding()` und `getAttributes()` für das eigene GUI-Element überschreiben, um das gewünschte Verhalten zu implementieren.

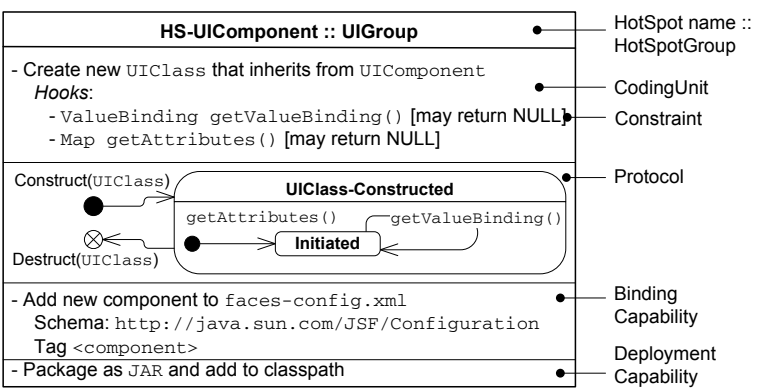

Abbildung 6: HotSpot HS-UIComponent in MyFaces 1.1.5

[2] http://myfaces.apache.org
[3] http://javaserverfaces.java.net/

Constraints der Hook-Methode `getAttributes()` besagen zusätzlich, dass diese Methode eine `Map` oder `NULL` zurückgeben muss.

Nehmen wir an, unsere fiktive Anwendung implementiert eigene GUI-Komponenten auf diese Weise. Die Benutzungsanalyse erkennt, dass es in der Anwendung Klassen gibt, die von `UIComponent` erben. Es wird erkannt, dass `UIComponent` zum Erweiterungspunkt `HSUIComponent` gehört. Nun verändern wir die Version von MyFaces 1.1.5 auf 1.2.0. In der neueren Version haben sich einige Dinge verändert, wie man dem Auszug aus der Dokumentation in Abbildung 7 entnehmen kann.

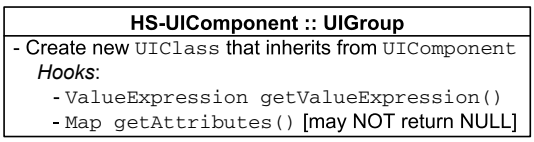

Abbildung 7: HotSpot HS-UIComponent in MyFaces 1.2.0

Wir führen eine Differenzanalyse durch und erhalten das Analyseergebnis in Tabelle 2. Es zeigt Ereignisse wie neue, gelöschte oder veränderte Elemente in der Dokumentation und einen Hinweistext zur späteren Auswertung. Wir sehen, dass wir in unseren GUI-Komponenten die Methode `getValueBinding()` nicht mehr verwenden dürfen und die neue Hook-Methode `getValueExpression()` zu berücksichtigen ist. Darüber hinaus können z.B. auch Änderungen der Constraints erkannt werden, so dass man nun weiß, dass die Hook-Methode `getAttributes()` in Version 1.2.0 nicht mehr den Wert `NULL` zurückgeben darf. Auch hier nicht gezeigte Veränderungen am Kommunikationsprotokoll zwischen Framework und Erweiterung können so identifiziert werden.

Ereignis	Hinweis
Neu	Der Erweiterungspunkt 'HS-UIComponent' verwendet nun die neue Hook-Methode 'getValueExpression()'.
Löschung	Der Erweiterungspunkt 'HS-UIComponent' verwendet die Hook-Methode 'getValueBinding()' nicht mehr'.
Änderung	Der Constraint der Hook-Methode 'getAttributes()' wurde geändert. *1.1.5*: 'Die Methode muss eine Map oder NULL zurückgeben.' *1.2.0*: 'Die Methode muss eine Map zurückgeben – nicht NULL.'

Tabelle 2: Ergebnis einer Differenzanalyse

Die Differenzanalyse ermittelt sämtliche Veränderungen an der Frameworkdokumentation. In einer abschließenden Kompatibilitätsanalyse würde man nun das Ergebnis der Differenzanalyse bezüglich verwendeter Erweiterungspunkte betrachten. Während dieser Analyse können die Aussagen hinsichtlich potentiell brechender oder brechender Einschränkungen bewertet werden. Die Löschung der Hook-Methode `getValueBinding()` ist zum Beispiel eine brechende Einschränkung. Eine Veränderung des Constraints ist potentiell brechend. Das Ergebnis der Kompatibilitätsanalyse ist eine bewertete Liste aller relevanten Veränderungen.

5 Implementierung

Wir setzen unseren Ansatz für Java-basierte Frameworks um, da Java eine häufig eingesetzte Technologie bei der Entwicklung betrieblicher Informationssysteme ist. Zur Erstellung einer FDMM-basierten Frameworkdokumentation wird das FDMM mit Hilfe des "Eclipse Modeling Frameworks" (EMF) [SBPM09] umgesetzt. Diese EMF-Umsetzung des FDMM ist die Grundlage für die Implementierung eines Editors, der sich in die Entwicklungsumgebung Eclipse integriert und so die parallele Dokumentation eines Frameworks während seiner Entstehung ermöglicht.

Zur Differenzanalyse planen wir das Werkzeug EMF-Compare[4] einzusetzen, das es ermöglicht die Unterschiede zwischen zwei EMF-basierten Modellen, wie dem FDMM, zu bestimmen. Die Benutzungsanalyse basiert auf dem Open Source Werkzeuge "Dependency Finder"[5], das eine statische Codeanalyse auf Java-Bytecode-Ebene durchführt. Durch Erweiterungen dieses Werkzeuges können die Codebezüge einer Anwendung zu verwendeten Java-Archiven (JAR-Dateien) in einer Datenbank gespeichert werden, so dass diese einmal erhobenen Informationen für weitere Analysen zur Verfügung stehen. Java-Archive sind hierbei das technische Format, in dem Frameworks ausgeliefert werden, so dass wir mit dem Dependency Finder Informationen über die Benutzungsbeziehung zwischen Anwendung und Framework erhalten. Für die Zukunft sind Erweiterungen der Benutzungsanalyse geplant, die in der Lage sein werden zusätzlich Informationen aus Ressourcen wie Konfigurationsdateien auszuwerten.

Zur Umsetzung der Kompatibilitätsanalyse setzt ein eigenes Werkzeug die gewonnenen Informationen aus der Benutzungsanalyse mit den Informationen der Differenzanalyse in Beziehung. Für jede Benutzung eines Erweiterungspunktes des Frameworks prüfen wir, ob sich laut Frameworkdokumentation etwas an der Umsetzung dieses Erweiterungspunktes verändert hat. Ist dies der Fall, wird die Veränderung entsprechend bewertet und eine Aussage zur Kompatibilitätseinschränkung generiert. Die Liste der ermittelten Einschränkungen in Form einer Textdatei bildet die Ausgabe der Kompatibilitätsanalyse.

6 Diskussion und Ausblick

Für die praktische Anwendbarkeit unseres Ansatzes arbeiten wir zunächst daran die beschriebene Werkzeugkette stabil zu implementieren und Erfahrungen durch praktische Evaluationen zu sammeln. Auf Forschungsseite wollen wir das hier beschriebene Problem zwischen einer Anwendung und einem Framework auf die Beziehung zwischen Anwendung und beliebig vielen Frameworks ausweiten. Eine komplexere Anwendung, skizziert in Abbildung 8, verwendet z.B. in jeder Anwendungsschicht spezialisierte Frameworks, die an ihren Erweiterungspunkten miteinander interagieren. Darüber hinaus verwendet jedes Framework eine Reihe von Bibliotheken, um z.B. XML-Dokumente verarbeiten zu können oder Log-Meldungen zu generieren. Hinzu kommen die Anteile eigener Erwei-

[4] http://wiki.eclipse.org/EMF_Compare
[5] http://depfind.sourceforge.net

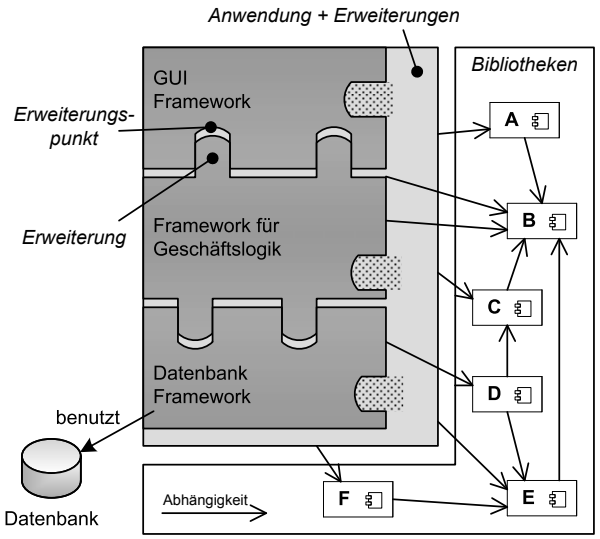

Abbildung 8: Skizze einer komplexen Anwendung

terungen der Frameworks plus benötigter eigener Bibliotheken, wobei auch Bibliotheken untereinander Abhängigkeiten haben können. Es entsteht ein komplexes Abhängigkeitsgeflecht, dass es bei der Aktualisierung eines Frameworks zu beachten gilt. Es gilt also die Kompatibilitätsanalyse auf mehrere Frameworks plus abhängiger Bibliotheken ausdehnen zu können. In diesem Zusammenhang planen wir Verbesserungen für die Integration von Benutzungs-, Differenz- und Kompatibilitätsanalyse in eine produktreifere Werkzeugkette, die als Open-Source-Software in Zukunft auch einmal in der industriellen Praxis Verwendung finden kann.

Literatur

[BBAGH03] N. Bouassida, H. Ben-Abdallah, F. Gargouri und A. B. Hamadou. Formalizing the framework design language F-UML. In *Proceedings of SEFM'03*, Seiten 164–172. IEEE, September 2003.

[CBE+10] Fabian Christ, Jan-Christopher Bals, Gregor Engels, Christian Gerth und Markus Luckey. A Generic Meta-Model-based Approach for Specifying Framework Functionality and Usage. In *Proceedings of TOOLS Europe'10*, Jgg. 6141 of *LNCS*, Seiten 21–40. Springer, 2010.

[Cla06] Program Compatibility Approaches. In Frank de Boer, Marcello Bonsangue, Susanne Graf und Willem-Paul de Roever, Hrsg., *Formal Methods for Components and Objects*, Jgg. 4111 of *LNCS*, Seiten 243–258. Springer, 2006.

[CZ06] D.C. Craig und W.M. Zuberek. Compatibility of Software Components - Modeling and Verification. In *Proceedings of the International Conference on Dependability of Computer Systems (DepCos-RELCOMEX'06)*, Seiten 11–18, May 2006.

[Deu89] L. P. Deutsch. Design reuse and frameworks in the Smalltalk-80 system. *Software reusability*, 2:57–71, 1989.

[FPR00] Marcus Fontoura, Wolfgang Pree und Bernhard Rumpe. UML-F: A Modeling Language for Object-Oriented Frameworks. In *Proceedings of the European Conference on Object-Oriented Programing (ECOOP2000), Sophia Antipolis and Cannes, France*, June 2000.

[FPR02] Marcus Fontoura, Wolfgang Pree und Bernhard Rumpe. *The UML Profile for Framework Architectures*. Addison Wesley, 2002.

[Gos68] John A. Gosden. Software compatibility: what was promised, what we have, what we need. In *Proceedings of the fall joint computer conference, part I (AFIPS '68)*, Seiten 81–87, New York, NY, USA, 1968. ACM.

[JF88] Ralph E. Johnson und Brian Foote. Designing Reusable Classes. *Journal of Object-Oriented Programming*, 1(2):22–35, June/July 1988.

[Joh97] Ralph E. Johnson. Frameworks = (Components + Patterns). *Commun. ACM*, 40(10):39–42, 1997.

[LSTM06] Srgio Lopes, Carlos Silva, Adriano Tavares und Joo Monteiro. Describing Framework Static Structure: promoting interfaces with UML annotations. In *Proceedings of the WCOP'06*, Seiten 54–61. Universität Karlsruhe - Fakultät für Informatik, July 2006. ISSN 1432 - 7864.

[MCK97] Matthias Meusel, Krzysztof Czarnecki und Wolfgang Köpf. A model for structuring user documentation of object-oriented frameworks using patterns and hypertext. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'97)*, Jgg. 1241 of *LNCS*, Seiten 496–510. Springer, 1997.

[ME04] Stephan McCamant und Michael D. Ernst. Early Identification of Incompatibilities in Multi-component Upgrades. In *Proceedings of the ECOOP'04, Oslo (Norway)*, Jgg. 3086 of *LNCS*, Seiten 117–140. Springer, 2004.

[OCM99] A. Ortigosa, M. Campo und R. Moriyn. Enhancing framework usability through smart documentation. In *Proceedings of the Argentinian Symposium on Object Orientation*, Seiten 103–117, Buenos Aires, Argentina, 1999.

[OMG06] OMG. Meta Object Facility (MOF) Core Specification, V2.0. Bericht, OMG, 2006.

[OMG10] OMG. Unified Modeling Language (OMG UML), Superstructure, V2.3, May 2010.

[Par11] David Lorge Parnas. Precise Documentation: The Key to Better Software. In Sebastian Nanz, Hrsg., *The Future of Software Engineering*, Seiten 125–148. Springer, 2011.

[SBPM09] Dave Steinberg, Frank Budinsky, Marcelo Paternostro und Ed Merks. *EMF - Eclipse Modeling Framework*. Addison Wesley, second edition. Auflage, 2009.

[Sch97] Han Albrecht Schmid. Systematic framework design by generalization. *Commun. ACM*, 40(10):48–51, 1997.

[SSP95] Albert Schappert, Peter Sommerlad und Wolfgang Pree. Automated support for software development with frameworks. In *Proceedings of the Symposium on Software reusability (SSR '95)*, Seiten 123–127, New York, NY, USA, 1995. ACM Press.