Studienarbeit

Rahmenwerkbasierte Werkzeugkomponenten am Beispiel des JWAM-Rahmenwerks

Fachbereich Informatik
Arbeitsbereich Softwaretechnik
Universität Hamburg
Vogt-Kölln-Straße 30
22527 Hamburg

August 2000

Betreuung: Heinz Züllighoven,
Stefan Roock

Johanna Felski
4724111
Karl-Rüther-Stieg 5
21035 Hamburg
4felski@informatik.uni-hamburg.de

Erdal Özkan
4724200
Karl-Rüther-Stieg 5
21035 Hamburg
4oezkan@informatik.uni-hamburg.de

Inhaltsverzeichnis

1 Einleitung und Überblick

1.1 Motivation

Der Begriff der Komponente und der komponentenorientierten Softwareentwicklung wird in den letzten Jahren immer häufiger benutzt. Dieser Ansatz verspricht Vorteile gegenüber der klassischen objektorientierten Softwareentwicklung. Ein Ziel der komponentenorientierten Softwareentwicklung ist die Steigerung der Produktivität des Softwareentwicklungsprozesses und der Qualität der Anwendungen. Bei großen und komplexen Softwaresystemen entsteht der Wunsch nach grobkörnigeren Modellierungseinheiten als Klassen, den Komponenten. Durch Komponenten hoffen die Anwendungsentwickler, die Komplexität der Systeme zu reduzieren. Eine lose Kopplung zwischen den Komponenten und eine striktere Umsetzung des Geheimnisprinzips als bei Klassen versprechen dem Anwendungsentwickler einen einfacheren Austausch von Komponenten, was eine bessere Anpassbarkeit der Anwendung zur Folge hätte. Die Komponenten sollen projektübergreifende Dienste zur Verfügung stellen, damit sie in möglichst vielen Kontexten wiederverwendet werden können.

Die Vision, die hinter der komponentenorientierten Softwareentwicklung steht, ist die Anwendungsentwicklung nach dem Baukastenprinzip. Hierbei soll die Anwendung aus einzelnen Komponenten durch den Anwendungsentwickler zusammengesetzt werden. Der Grossteil der Aufgaben des Anwendungsentwicklers bestünde in der geschickten Auswahl und dem Zusammensetzen der Komponenten. Auf der anderen Seite würde es spezialisierte Komponentenentwickler geben, die sich auf einen Bereich konzentrieren und qualitativ hochwertige Komponenten anbieten.

1.2 Ziele der Arbeit

In dieser Studienarbeit möchten wir uns auf Komponenten im Rahmen der Werkzeugentwicklung nach WAM beschränken. Die Grundlage für den von uns verwendeten Komponentenbegriff hat Fröse (vgl.[Fröse99]) in seiner Diplomarbeit erarbeitet.

Durch den Einsatz von Komponenten in der Werkzeugentwicklung erhoffen wir uns einerseits eine bessere Wiederverwendbarkeit vorhandener Werkzeuge, andererseits die Vereinfachung der Werkzeugkonstruktion.

Im Rahmen dieser Arbeit wollen wir ein Konzept der Werkzeugkomponenten entwickeln. Dieses Konzept soll für das JWAM-Rahmenwerk umgesetzt werden.

1.3 Übersicht über die Arbeit

Im Kapitel **Rahmenwerke und Komponenten** betrachten wir die Begriffe Komponente und Rahmenwerk. Wir erläutern die Vorteile, die sich aus der Verwendung der komponentenorientierten Softwareentwicklung mit Rahmenwerken ergeben.

Im Kapitel **Werkzeugkonstruktion in JWAM** stellen wir das JWAM-Rahmenwerk und seine Rolle bei der bisherigen Werkzeugentwicklung nach dem

WAM-Ansatz vor. Außerdem führen wir hier ein begleitendes Beispiel-Werkzeug ein.

Das Kapitel **Werkzeugkomponenten** zeigt das im Rahmen dieser Studienarbeit erstellte Konzept der Werkzeugkomponenten. Wir beschreiben die einzelnen Elemente der zugrundeliegenden Modellarchitektur.

Im Kapitel **Realisierung in JWAM** zeigen wir die Umsetzung des vorgestellten Konzeptes für das JWAM-Rahmenwerk.

Das letzte Kapitel **Abschluss und Ausblick** fasst die Ergebnisse dieser Arbeit zusammen und gibt einen Ausblick in die mögliche Weiterentwicklung.

1.4 Danksagung

An dieser Stelle möchten wir allen danken, die bei der Entstehung dieser Arbeit mitgewirkt haben. Insbesondere danken wir Stefan Roock, der uns mit seinen Ratschlägen und seiner Hilfe immer zur Seite stand.

2 Rahmenwerke und Komponenten

In diesem Kapitel beschreiben wir, was wir unter den Schlagwörtern komponentenorientierte Softwareentwicklung und Komponenten verstehen, und welche Vorteile uns die Anwendung dieser Techniken gegenüber der klassischen Objektorientierung bieten. Die komponentenorientierte Softwareentwicklung verspricht Vorteile gegenüber der klassischen Objektorientierung, wie die Erhöhung der Wiederverwendbarkeit einzelner Teilstrukturen und die Vereinfachung der Anwendungsentwicklung. Für große Softwaresysteme ist der Einsatz von Rahmenwerken unerlässlich. Rahmenwerke bilden ein architektonisches Gerüst, reduzieren den Aufwand und Erhöhen die Qualität großer Softwaresysteme. Allerdings ist die Entwicklung und der Einsatz eines großen Rahmenwerks komplex.

2.1 Komponentenorientierte Softwareentwicklung

Nach Szyperski (vgl. [Szyperski98]) kann die traditionelle Softwareentwicklung im allgemeinen in zwei Lager aufgeteilt werden. Auf der einen Seite steht die klassische Anwendungsentwicklung. Hier wird ein Projekt von Anfang an mit Hilfe von Programmierwerkzeugen und Bibliotheken erzeugt. Auf der anderen Seite steht das Erwerben und Anpassen von Standardsoftware. Die Standardsoftware bietet für einen bestimmten Anwendungsfall eine allgemeine Lösung. Der Anwendungsentwickler versucht durch Parametrisierung die Standardsoftware so nah wie möglich an die erwünschte Lösung zu bringen. Beide Vorgehensweisen haben Vor- und Nachteile. Der Hauptvorteil bei einer Anwendungsentwicklung von Anfang an ist die optimale Anpassung an das Geschäftsmodell des Kunden. Der Nachteil bei einer solchen Vorgehensweise liegt im ökonomischem Bereich. Die zu modellierenden Strukturen sind bei größeren Projekten sehr komplex, was zu langen Entwicklungszeiten führt. Die langen Entwicklungszeiten stellen ein Problem dar, weil sie auf der einen Seite hohe Kosten verursachen und auf der anderen Seite eine termingerechte Fertigstellung der Software gefährden. Der Vorteil bei der Standardsoftware liegt im begrenzten finanziellen Risiko. Der Kauf einer Standardsoftware ist günstiger als die Neuentwicklung. Der Nachteil der Software liegt in der Anpassbarkeit und in der Flexibilität. Die Standardsoftware lässt sich nur begrenzt an die Bedürfnisse der Kunden anpassen. Die Folge ist, dass die Software nicht an die Anforderungen der Kunden anpasst wird, sondern die Kunden ihre Geschäftsprozesse an die Standardsoftware anpassen müssen. Diese Schwierigkeiten verstärken sich, wenn sich das Geschäftsmodell des Kunden ändert. Nachdem Szyperski diese beiden Extreme der Softwareentwicklung skizziert, zeigt er die Softwareentwicklung mit Komponenten als einen Mittelweg auf. Eine Komponentensoftware ist ein System aus zusammengesetzten Komponenten. Jede Komponente ist nach Szyperski vergleichbar mit Standardsoftware und deren Vorteilen. Wobei das Zusammenstellen der Komponenten die Erwünschte Anpassbarkeit bringt.

Das Ziel der Komponentenorientierung ist eine Softwareentwicklung nach dem sogenannten Baukastenprinzip. Sie beschreibt ein Zusammensetzen bereits vorhandener, von Drittherstellern erworbener oder selbstentwickelter Softwarebausteine zu einer komplexen Anwendung. Idealerweise könnte ein

entsprechendes Entwicklungswerkzeug den Anwendungsentwickler unterstützen und die oben beschriebene Komposition und Auswahl der Komponenten erleichtern. Die einzelnen Komponenten sollen durch Parametrisierung an den Anwendungskontext angepasst werden.

Wie Fröse (vgl. [Fröse99]) in seiner Diplomarbeit beschreibt, setzen wir die komponentenorientierte Softwareentwicklung als Abstraktionsmittel auf der Softwarearchitekturebene ein und unterscheiden so architekturspezifische und implementierungsspezifische Aspekte. Wir erhoffen uns damit, komplexe Softwaresysteme, vor allem für den Anwendungsentwickler, einfacher und übersichtlicher zu gestalten.

Wir teilen die Auffassung von Griffel (vgl. [Griffel98]), dass die Komponentenorientierung und die Objektorientierung keine nebeneinanderstehenden, konkurrierenden Technologien sind, sondern aufeinander aufbauen. Griffel versteht die Komponentenorientierung als Skalierung der Objektorientierung. Aus unserer Sicht bilden die Objektorientierung und Rahmenwerke die Grundlage für Komponenten. Unser Verständnis von Komponentenorientierung unterscheidet sich in diesem Punkt von der Auffassung Szyperskis. Für Szyperski ist die interne Implementierung einer Komponente nicht entscheidend.

2.1.1 Nutzen der komponentenorientierten Softwareentwicklung

Die Erfahrungen im Bereich der Rahmenwerke zeigen, dass die Komplexität eines Rahmenwerks schnell zunimmt, und somit von einer einzelnen Person kaum noch zu überschauen ist. In einer Komponente wird eine Funktionalität, die vorher in mehreren kooperierenden Klassen verteilt war, gekapselt. Durch die Strukturierung eines Rahmenwerks mit Komponenten wird die Komplexität reduziert, so dass das Erlernen und die Überschaubarkeit des Rahmenwerks einfacher wird.

Die Komponentenorientierung verspricht als Hauptvorteil die Wiederverwendung. Dabei möchte der Entwickler sowohl eigene Komponenten aus früheren Projekten als auch Komponenten eines Drittherstellers einsetzen. Durch solche Wiederverwendung der Software ist eine effizientere Entwicklung möglich.

Griffel (vgl. [Griffel98]) verdeutlicht in diesem Zusammenhang, dass für eine „gute“ Komponente, die sich durch den hohen Wiederverwendungsgrad auszeichnet, ein projektübergreifendes Denken nötig ist. Der Entwurf einer solchen Komponente, die sich in mehreren Kontexten wiederverwenden lässt, ist schwerer als eine Komponente zu entwerfen, die nur für ein Anwendungsfall konzipiert ist. Auf der einen Seite gestaltet sich die Anwendungsentwicklung durch Zusammensetzen von Komponenten gegenüber der klassischen Objektorientierung einfacher. Auf der anderen Seite sieht man sich der schwierigen Aufgabe gegenüber, „gute“ Komponenten zu entwickeln, die sich auf die oben beschriebene Weise in mehreren Anwendungskontexten wiederverwenden lassen.

Durch Wiederverwendung einer bewährten, in einem Anwendungskontext ausgetesteten und eingesetzten Komponente erhöht sich deren Qualität, und somit die Qualität der Anwendung insgesamt.

2.1.2 Zum Begriff der Komponente

In der Literatur wird der Begriff der Komponente unterschiedlich verwendet. Ein wichtiger Aspekt in beinahe allen vorhandenen Definitionen ist die Wiederverwendbarkeit von Komponenten und die wohldefinierten Schnittstellen, über die eine Komponente ihre Dienste zur Verfügung stellt, sowie Dienste anderer Komponenten in Anspruch nimmt. Bäumer (vgl. [Bäumer98]) stellt die fachliche Funktionalität der Komponente als ihre weitere wichtige Eigenschaft heraus.

Szyperski definiert eine Komponente wie folgt:

> *„A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.“ ([Szyperski98], Seite 34)*

Szyperski sieht einen potentiellen Komponentenmarkt, in dem Komponenten als fertige, unabhängige Einheiten erworben und durch Zusammensetzen sowie Parametrisieren an den Anwendungsfall angepasst werden können.

In seinem Verständnis des Begriffs der Komponente definiert Fröse:

> *„Eine Komponente ist eine benannte, (wieder)verwendbare Einheit des Architekturmodells, die innerhalb eines Softwaresystems definierte fachliche Dienstleistungen erbringt. Eine Komponente besitzt eine Menge explizit definierter und benannter Schnittstellen, über die sie mit ihrer Umwelt interagiert und hinter denen Details der Komponentenimplementation verborgen bleiben. [...] Jede Komponente ist Exemplar einer Komponentenklasse.“([Fröse99], S. 7)*

Die Definition von Fröse zeigt, dass eine Komponente ein Bestandteil des Architekturmodells ist und sich nicht auf eine technische Sicht reduzieren lässt. Weiterhin ist die Unterscheidung zwischen der Laufzeitversion einer Komponente und ihrer statischen Repräsentation wichtig.

Zusammenfassend stellen wir fest:

> Eine Komponente ist grobkörniger als eine Klasse. Sie kapselt mehrere kooperierende Klassen, so dass auf deren Schnittstellen nicht mehr direkt, sondern über eine einheitliche Schnittstelle zugegriffen wird. Über diese wohldefinierte Schnittstelle bietet eine Komponente ihre Dienste an und nimmt Dienste anderer Komponenten in Anspruch.

2.2 Rahmenwerke

Die Hoffnungen, die an Rahmenwerken geknüpft werden, sind ähnliche, wie die der komponentenorientierten Anwendungsentwicklung. Bei der Entwicklung großer Softwaresysteme erhofft man sich durch den Einsatz eines Rahmenwerkes gesteigerte Produktivität, kürzere Entwicklungszeiten und höhere Qualität der Anwendungssoftware.

Ein Rahmenwerk besteht aus Klassen, die im Gegensatz zu Modulbibliotheken nicht als ungeordnete Sammlung von Bausteinen vorliegen, sondern in einer flexiblen Hierarchie, aus erweiterbaren Bausteinen und Konzepten, stehen (vgl. [Züllighoven98]). Ein Rahmenwerk ist eine Architektur aus Klassenhierarchien. Griffel (vgl. [Griffel98]) spricht von einem konzeptuellen Rahmen, in dem eine

Beschreibung allgemeiner Leitlinien und Regeln festgehalten sind. Diese Rahmenwerkarchitektur bietet uns allgemeine generische Lösungen für ähnliche Probleme aus einem bestimmten Anwendungskontext. Züllighoven spricht von einem Anwendungsrahmenwerk, wenn das Rahmenwerk für einen fachlich eingegrenzten Anwendungsbereich konzipiert ist. Ein Rahmenwerk gibt den Kontrollfluss einer Anwendung vor, indem eine Konstruktion aus zusammenspielenden Klassen, und nicht eine einzelne Klasse, wiederverwendet wird. Die konkrete Anwendung wird durch Spezialisierung bestimmter Klassen oder durch Parametrisierung mit Hilfe vorgegebener Parameterobjekte erstellt. Ob die Anwendung überwiegend durch Spezialisierung oder Parametrisierung erzeugt wird, hängt von der Konzeption des Rahmenwerkes als Black-Box oder White-Box ab.

2.2.1 Black-Box-Verwendung

In Black-Box-Verwendung werden Exemplare aus „*gebrauchsfertigen*“ (vgl. [Pree97]) Rahmenwerksklassen verwendet. Darunter sind Klassen zu verstehen, die keine abstrakten Methoden besitzen und durch Exemplarbildung einsetzbar und lauffähig sind. Sie werden erzeugt und durch Konfiguration an den Anwendungskontext angepasst. Die Konfiguration erfolgt in der Regel durch Einsetzen von Konfigurationsobjekten in dafür vorgesehene Einschübe. Die Black-Box-Verwendung eines Rahmenwerks ist in Abbildung 1 verdeutlicht. In der Abbildung haben wir zwei Vererbungshierarchien dargestellt. Auf der einen Seite ist eine abstrakte Klasse, die eine Schnittstelle implementiert, abgebildet. Von dieser abstrakten Klasse wird eine konkrete Klasse abgeleitet, die keine abstrakten Methoden mehr hat, also „*gebrauchsfertig*“ ist. Auf der anderen Seite gibt es eine Konfigurationsklasse, die von einer Konfigurationsschnittstelle abgeleitet ist. Mit dieser Konfigurationsklasse kann man die konkrete Klasse parametrisieren. Der Anwendungsentwickler kann ohne Kenntnis von Interna des Rahmenwerks, Exemplare von der konkreten Klasse und von der Konfigurationsklasse erzeugen. Das Objekt vom Typ „konkrete Klasse“ wird mit dem Objekt der Konfigurationsklasse parametrisiert.

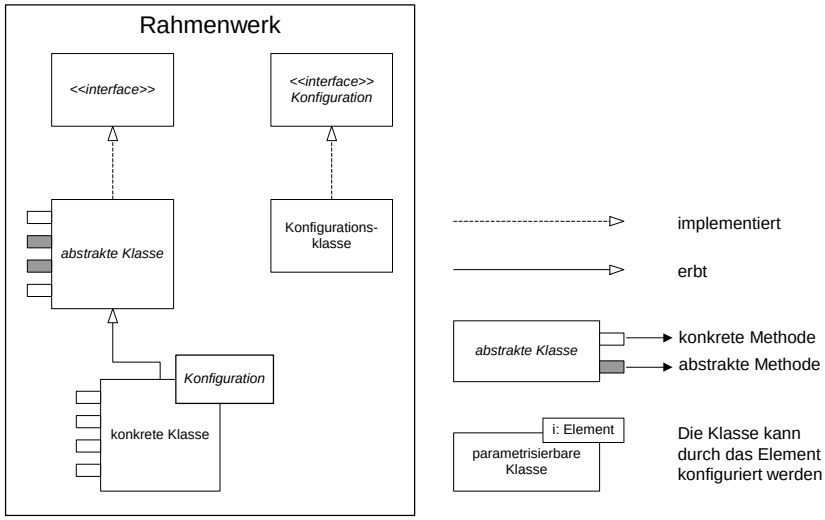

Abbildung 1: Black-Box-Verwendung

2.2.2 White-Box-Verwendung

Bei White-Box-Verwendung werden Unterklassen von speziell dafür vorgesehenen abstrakten Klassen gebildet. Um das Verhalten des Rahmenwerks zu verändern, werden Methoden in den Unterklassen überschrieben. Pree (vgl. [Pree97]) nennt diese Methoden Einschubmethoden, durch die das Rahmenwerk angepasst werden kann.

Die White-Box-Verwendung haben wir in Abbildung 2 dargestellt. Der Anwendungsentwickler leitet eine Klasse von einer bestehenden Rahmenwerksklasse ab, um neue Funktionalität implementieren zu können. Dabei muss er in der Regel eine abstrakte Klasse spezialisieren. Der Anwendungsentwickler ist dafür verantwortlich, die abstrakten Methoden zu konkretisieren, wofür er Kenntnisse über den internen Aufbau und die Realisierung der Rahmenwerksklassen benötigt.

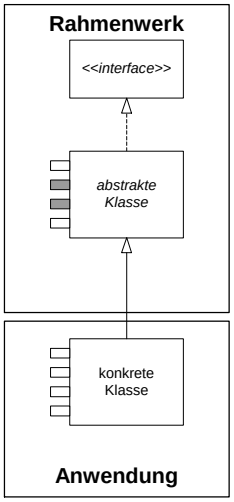

Abbildung 2: White-Box-Verwendung

Die neu entwickelten Klassen können auf der einen Seite für die Anwendung benutzt werden und auf der anderen Seite das Rahmenwerk ergänzen. Die White-Box-Verwendung eines Rahmenwerks erfordert vom Anwendungsentwickler das Verständnis über dessen Entwurf und die Implementierung, was in der Regel bei großen und komplexen Rahmenwerken ein Problem darstellt. Pree bezeichnet Black-Box-Rahmenwerke als Endstadien einer Entwicklung aus White-Box-Rahmenwerken (vgl. [Pree97]). Wie in [Züllighoven98] beschrieben, gibt es keine starre Unterscheidung zwischen Black-Box- und White-Box-Rahmenwerken. Vielmehr ergibt sich die Einteilung aus dem Verwendungszusammenhang.

2.3 Entwicklungsprozess mit Rahmenwerken und Komponenten

In der folgenden Tabelle haben wir stichwortartig die Vor- und Nachteile der komponentenorientierten Softwareentwicklung und der Anwendungsentwicklung mit Rahmenwerken gegenüber der klassischen Objektorientierung dargestellt.

Komponentenorientierte Softwareentwicklung	Anwendungsentwicklung mit Rahmenwerken
Vorteile gegenüber klassischer Objektorientierung:	
bessere Wiederverwendbarkeit	bessere Wiederverwendbarkeit
höhere Qualität der Software	höhere Qualität der Software
gesteigerte Produktivität	gesteigerte Produktivität
kürzere Entwicklungszeiten	kürzere Entwicklungszeiten
Reduzierung der Komplexität	architektonischer Rahmen
Nachteile gegenüber klassischer Objektorientierung:	
Liegen als ungeordnete Sammlung von Softwarebausteinen vor; Keine gemeinsame Semantik und kein architektonischer Rahmen.	Große Rahmenwerke sind komplex in der Entwicklung und in der Pflege.

Tabelle 1: Vor- und Nachteile der komponentenorientierten Softwareentwicklung und der Anwendungsentwicklung mit Rahmenwerken gegenüber klassischer Objektorientierung

Die Gegenüberstellung lässt die Schlussfolgerung zu, die Komponentenorientierung und die Anwendungsentwicklung mit Rahmenwerken zu kombinieren. Die erhofften Vorteile der Komponentenorientierung decken sich zum größten Teil mit den Vorteilen der Anwendungsentwicklung mit Rahmenwerken. Die Komplexität bei dem Entwickeln und Einsetzen von großen Rahmenwerken kann mit Hilfe von Komponenten verringert werden. Rahmenwerke können den Komponenten den benötigten architektonischen Rahmen bieten.

Wir halten fest:

> Die Komponentenorientierung und die Anwendungsentwicklung mit Rahmenwerken auf Basis der Objektorientierung können sich gegenseitig Ergänzen. Auf der einen Seite bietet das Rahmenwerk den Komponenten den benötigten architektonischen Rahmen. Auf der anderen Seite wird die Komplexität beim Einsatz und der Pflege des Rahmenwerks durch den Einsatz von Komponenten reduziert.

Es gibt unterschiedliche Visionen, die hinter der Anwendungsentwicklung mit Rahmenwerken und der Anwendungsentwicklung mit Komponenten stehen. In der Anwendungsentwicklung mit Komponenten bedient sich der Entwickler aus einem Pool von vollständigen, in sich abgeschlossenen funktionalen Einheiten, den Komponenten (vgl. [Griffel98]). Diese werden durch eine geschickte Auswahl,

Konfiguration und Verknüpfung zu einer Anwendung zusammengefügt. Die Idealvorstellung ist dabei die Unterstützung dieser Aufgabe durch ein geeignetes Werkzeug, so dass die Produktivität der Anwendungsentwicklung gesteigert werden kann.

Die Anwendungsentwicklung mit Rahmenwerken sieht so aus, dass das Rahmenwerk die Grundstruktur und den konzeptuellen Rahmen vorgibt. Der Anwendungsentwickler verfeinert diese Strukturen und bindet sie in seine Anwendungen ein. Wie wir weiter oben deutlich gemacht haben, bilden die Objektorientierung und Rahmenwerke die Grundlagen für eine erfolgreiche Komponentenorientierung. Auf Rahmenwerken basierende Komponenten sind im Unterschied zu Komponenten, die als ungeordnete Sammlung von Softwarebausteinen vorliegen, in eine gemeinsame Semantik und architektonische Sicht eingebettet. Die Identifizierbarkeit und die Entscheidung, ob die Komponente für den geplanten Einsatz angemessen ist, ist mit dem Rahmenwerk einfacher zu treffen. Auf der anderen Seite können die Rahmenwerke selbst die Vorteile der Komponenten nutzen und Bereiche vorsehen, in denen Komponenten eingeschoben werden können. Die Komponentenorientierung bietet im Bereich der Rahmenwerksentwicklung die gleichen Vorteile, wie bei der Anwendungsentwicklung. Wie oben beschrieben, kann durch eine Strukturierung mit Komponenten die Funktionalität aus mehreren kooperierenden Klassen gekapselt werden, wodurch eine Reduzierung der Komplexität erreicht wird. Die Komplexitätsreduktion wirkt sich vor allem für den Anwendungsentwickler beim Erlernen und Anwenden des Rahmenwerks positiv aus.

Griffel (vgl. [Griffel98]) sieht ein Problem in dem Reifeprozess, den ein Rahmenwerk durchläuft. Die Rahmenwerkentwicklung sei kein abgeschlossener Prozess, sondern durchlaufe eine evolutionäre Entwicklung. Das Problem entsteht, wenn die anfangs entwickelten Anwendungen ständig an das Rahmenwerk angepasst werden, und somit mit der Entwicklung des Rahmenwerks Schritt halten müssen. Wie Griffel beschreibt, könnte die komponentenorientierte Anwendungsentwicklung hier eine Lösung bieten, indem eine neue Generation von Komponenten eingesetzt wird, während die alten weiterhin ihren Dienst erbringen.

In Abbildung 3 veranschaulichen wir, wie wir uns eine solche Versionierung vorstellen. Wie in Abbildung 3a) zu sehen ist, kann eine Komponente eine zweite Version der Schnittstelle implementieren, während die erste Schnittstelle weiterhin als Dienst angeboten wird. In Abbildung 3b) wird eine zweite Komponente eingeführt, die die neue Version der Schnittstelle implementiert, während die erste Version aus Kompatibilitätsgründen weiterhin ihren Dienst erfüllt.

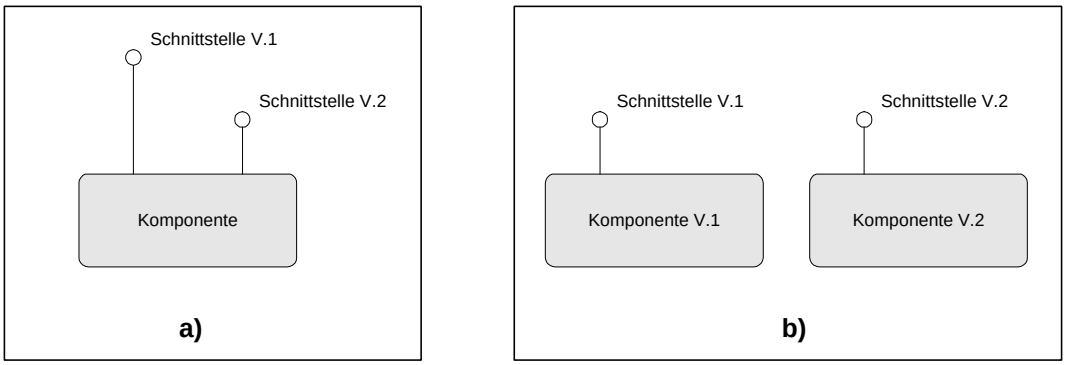

Abbildung 3: Versionierung von Komponenten

Anwendungsentwickler können von solch einer Vorgehensweise profitieren, da sie Anwendungen, die sich auf alte Schnittstellen abstützen nicht anzupassen brauchen. Anwendungen können somit schrittweise an Änderungen der Komponenten anpasst werden. Das oben beschriebene Problem bleibt unserer Meinung nach weiterhin bestehen, wenn wir mehrere Komponenten haben, die miteinander kooperieren. Die Kooperation stützt sich auf die Zustände der Komponenten ab. Änderungen am Zustandsmodell bewirken auch Änderungen an der Schnittstelle. In diesen Fällen müssen für eine Anwendung, die aus kooperierenden Komponenten besteht, die jeweiligen Komponenten angepasst werden, sonst können Typprobleme auftreten. Wir sind der Auffassung, dass mit zunehmender Größe des Systems ein weiteres Problem auftaucht. Durch die Vielzahl an Komponenten und deren Schnittstellenversionen kann eine Unüberschaubarkeit über die vorhandenen Schnittstellenversionen und deren Zusammenhänge mit anderen Komponenten entstehen.

2.4 Zusammenfassung

In diesem Kapitel haben wir unser Verständnis einer komponentenorientierten Softwareentwicklung und ihre Beziehung zur klassischen objektorientierten Anwendungsentwicklung mit Rahmenwerken dargelegt.

Die Objektorientierung und Rahmenwerke bilden die Grundlage für komponentenorientierte Softwareentwicklung. Durch Wiederverwendung einer bewährten, in einem Anwendungskontext ausgetesteten und eingesetzten Komponente, erhöht sich deren Qualität, und somit die Qualität der Anwendung insgesamt. Durch die Abstraktion von mehreren miteinander kooperierenden Klassen in Komponenten wird die Komplexität reduziert, so dass das Erlernen und die Überschaubarkeit des Rahmenwerks für Anwendungsentwickler einfacher wird.

Komponentenorientierung und die Anwendungsentwicklung mit Rahmenwerken auf Basis der Objektorientierung ergänzen sich gegenseitig. Auf der einen Seite bietet das Rahmenwerk den Komponenten den benötigten architektonischen Rahmen. Auf der anderen Seite wird die Komplexität beim Einsatz und der Pflege des Rahmenwerks durch die Benutzung von Komponenten reduziert.

3 Werkzeugkonstruktion in JWAM

Das JWAM-Rahmenwerk ist ein Applikations-Rahmenwerk nach dem WAM-Ansatz (vgl. [Züllighoven98]). Das Rahmenwerk bietet generische Lösungen für die Entwicklung interaktiver Anwendungen.

In diesem Kapitel möchten wir das JWAM-Rahmenwerk in der Version 1.4 und seine Rolle bei der Werkzeugentwicklung vorstellen.

3.1 Das JWAM-Rahmenwerk

JWAM ist ein Applikations-Rahmenwerk für die Entwicklung interaktiver Anwendungen in Java nach dem WAM-Ansatz. Das Rahmenwerk realisiert, wie in Abbildung 4 zu sehen, unterschiedliche Schichten. In den einzelnen Schichten werden thematisch zusammengehörige Bereiche zusammengefasst. Das JWAM-Rahmenwerk realisiert im JWAM-Kern die Sprach-, System-, Technologie- und Handhabungsschicht.

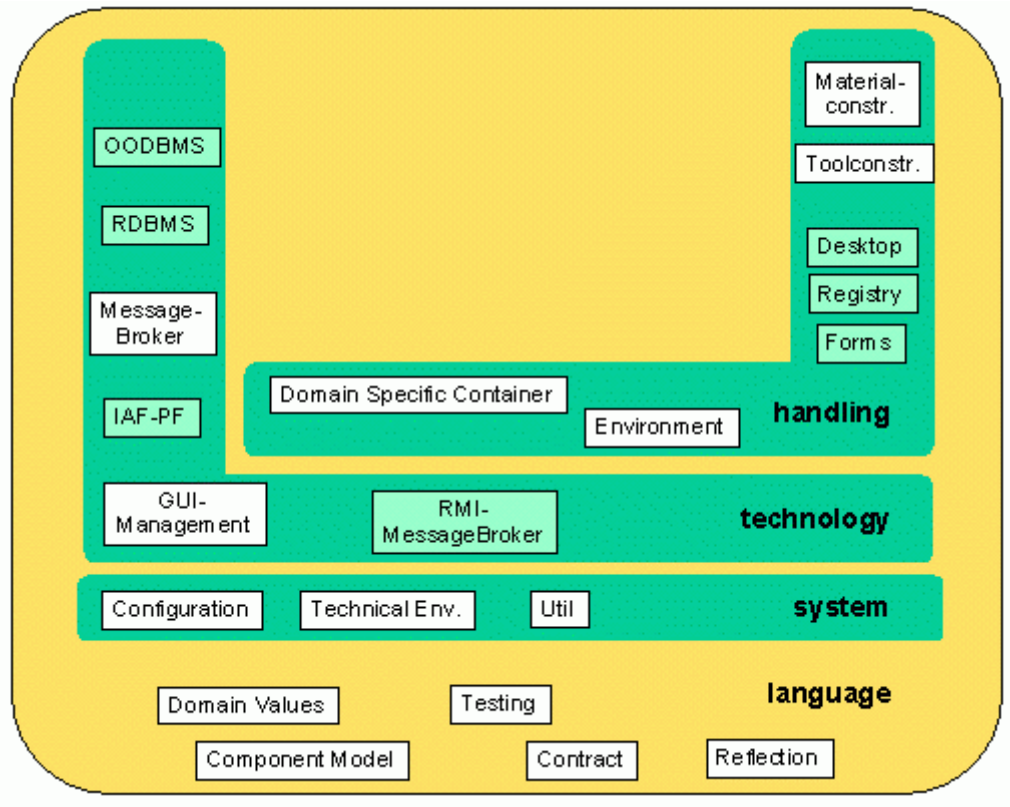

Abbildung 4: Die Kernfunktionalität der JWAM Systemarchitektur (aus [JWAM2000])

Die unterste Schicht des Rahmenwerkes bildet die Sprachschicht. In ihr sind Erweiterungen für die Sprache Java angesiedelt. Diese Erweiterungen werden im ganzen JWAM-Rahmenwerk benutzt und sind auch für den Anwendungsentwickler interessant. In dieser Schicht ist beispielsweise das Vertragsmodell nach Meyer (vgl. [Meyer97]) realisiert.

Die Systemschicht bietet verschiedene Abstraktionen über das darunterliegende Java-System an. Diese Schicht kapselt die technischen Komponenten eines Systems. Das Hauptziel der Systemschicht ist eine Abstraktion gegenüber Änderungen der Technologien zu schaffen, um somit eine stabile Basis für anwendungsfachlich nähere Schichten zu bilden.

In der Technologieschicht sind allgemeine Konzepte und Abstraktionen der verwendeten Technik angesiedelt. Die hier implementierten Komponenten werden in verteilten, objektorientierten Anwendungen gebraucht. Dazu gehört beispielsweise der Nachrichtenvermittler (Message-Broker). Die Technologieschicht benutzt insbesondere die Komponenten der Systembasisschicht.

Die Handhabungs- und Präsentationsschicht unterstützt die Entwicklung von Werkzeugen, Automaten und Materialien, indem sie grundlegende Mechanismen zu deren Konstruktion zur Verfügung stellt. In dieser Schicht werden die wiederverwendbaren Teile der Handhabung, der Funktionalität eines Werkzeugs, sowie die Präsentation an der Benutzungsoberfläche implementiert. Diese Studienarbeit beschäftigt sich daher mit dieser Schicht.

3.2 Werkzeugentwicklung mit WAM

Werkzeuge, wie sie im Rahmen des WAM-Konzepts verstanden werden, ermöglichen durch die Interaktion mit dem Benutzer das Verändern und Sondieren von Materialien. Dabei setzt sich ein Werkzeug (vgl. [Züllighoven98]) aus einer Interaktions- und einer Funktionskomponente zusammen.

Das WAM-Entwurfsmuster *Trennung Interaktion und Funktion* (Abbildung 5) beschreibt den prinzipiellen Werkzeugentwurf. Die fachliche Funktionalität des Werkzeugs (Funktionskomponente, Abk. FK) wird von dessen Präsentation an der Benutzungsoberfläche (Interaktionskomponente, Abk. IAK) getrennt. Die Funktionskomponente besitzt kein Wissen über ihre Interaktionskomponente. Im Gegensatz dazu kennt und benutzt die Interaktionskomponente ihre Funktionskomponente. Die strikte Trennung von Handhabung und Präsentation ermöglicht Änderungen oder sogar den Austausch der Interaktionskomponente, ohne das die Funktionskomponente angepasst werden muss. Die Rückkopplung von der Funktionskomponente an ihre Interaktionskomponente erfolgt über einen Ereignis- oder Beobachtermechanismus. Die Verbindung zwischen der Interaktionskomponente und der Funktionskomponente wird erst zur Laufzeit hergestellt, indem sich die Interaktionskomponente als Beobachter für bestimmte Ereignisse der Funktionskomponente anmeldet. Durch das Auslösen eines Ereignisses informiert die Funktionskomponente die Interaktionskomponente über ihre Zustandsänderungen. Die Interaktionskomponente kann auf die Zustandsänderungen entsprechend reagieren und die Darstellung des Materials aktualisieren. Das WAM-Muster *Rückkopplung zwischen Funktion und Interaktion* beschreibt diesen Zusammenhang.

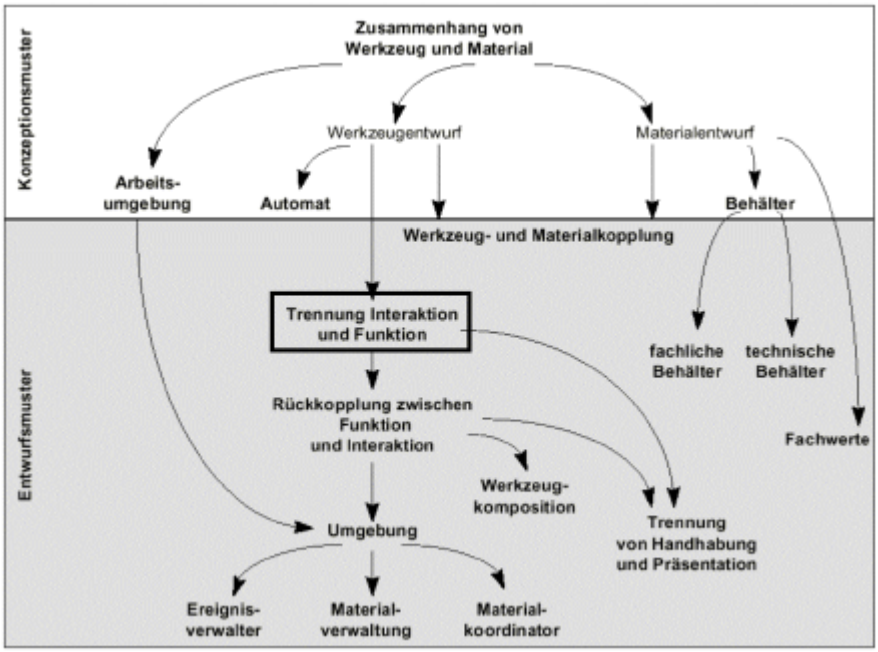

Abbildung 5: WAM-Muster (aus [Züllighoven98])

Für die Werkzeugkomposition bietet der WAM-Ansatz mehrere Entwurfsmuster an. Diese Muster beschreiben Lösungen für die Komposition von komplexen Werkzeugen aus einfachen Werkzeugen. Ein einfaches Werkzeug besitzt softwaretechnisch keine Sub-Werkzeuge. Im Gegensatz dazu ist ein Kombi-Werkzeug aus mehreren Sub-Werkzeugen zusammengesetzt. Die lose Kopplung zwischen den Teilen eines Kombi-Werkzeugs wird zum einen, wie im Muster *Rückkopplung zwischen Funktion und Interaktion*, über den Beobachtermechanismus und zum anderen über die Zuständigkeitskette (vgl. [Gamma96]) realisiert.

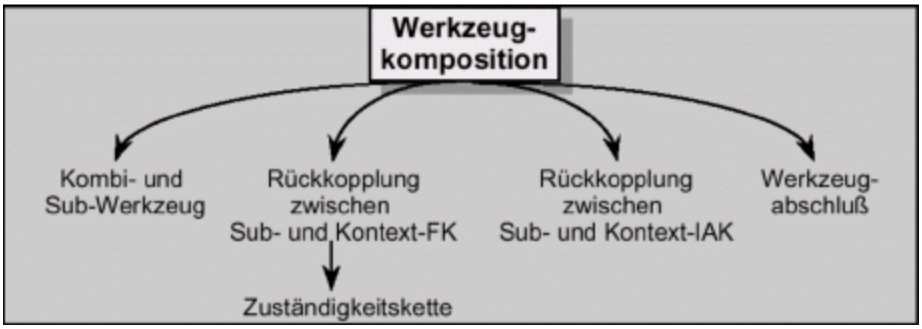

Abbildung 6: Werkzeugkomposition (aus [Züllighoven98])

Komplexe Werkzeuge aus nur einer Funktionskomponente und einer Interaktionskomponente aufzubauen, ist softwaretechnisch wenig sinnvoll. Zum einen würden die Werkzeuge mit wachsender Komplexität immer unüberschaubarer, zum anderen müsste man sie jedesmal von Grund auf neu implementieren.

Es ist also sinnvoll, komplexe Werkzeuge aus kleineren Werkzeugen, die eine elementare fachliche Funktionalität erfüllen, lose gekoppelt zu komponieren. Dadurch wird die Komplexität reduziert und die Wiederverwendung erhöht.

Das WAM-Muster *Kombi- und Sub-Werkzeug* beschreibt die obige Lösung. Solch ein Kombi-Werkzeug besteht aus mehreren Sub-Werkzeugen, die die fachlichen Teilaufgaben realisieren. Im softwaretechnischen Sinne ist ein Sub-Werkzeug in ein Kontext-Werkzeug eingebettet. In Abbildung 7 haben wir den Aufbau eines Kontext-Werkzeugs dargestellt. Bei einem Kombi-Werkzeug benutzen die Kontext-Funktionskomponente und die Kontext-Interaktionskomponente die jeweilige Funktions- bzw. Interaktionskomponente des Sub-Werkzeugs.

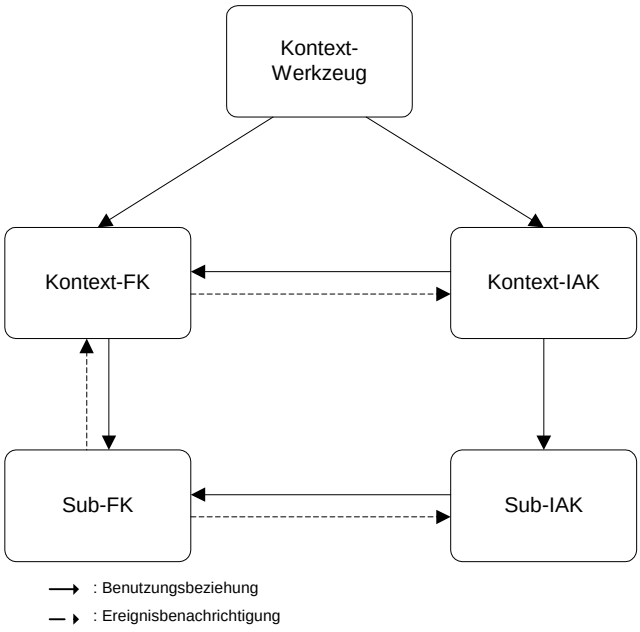

Abbildung 7: Kombi-Werkzeug mit einem Sub-Werkzeug

In einem Kombi-Werkzeug delegiert die Kontext-Funktionskomponente bestimmte Teilaufgaben an eine oder mehrere Sub-Funktionskomponenten. Das WAM-Entwurfsmuster *„Rückkopplung zwischen Sub- und Kontext-FK“* beschreibt eine Lösung für das Problem der Rückkopplung zwischen den Sub-FKs an die Kontext-FK. Die Kontext-FK muss die fachliche Schnittstelle der Sub-FKs kennen, um Teilaufgaben an die Sub-FKs weiterleiten zu können. Ähnlich wie bei der Rückkopplung zwischen Funktionskomponente und Interaktionskomponente wird die Kontext-FK über Zustandsänderungen der Sub-FKs mit Hilfe des Ereignis- oder des Beobachtermusters informiert. Dazu meldet sich die Kontext-FK, wie auch die IAKs, für die relevanten Ereignisse an. Ereignisse werden bei Zustandsänderungen von der Sub-FK an ihre Kontext-FK sowie die IAKs verschickt.

Die Anforderungen (*requests*) der Sub-FKs, die sie selbst nicht erfüllen können, werden an die Kontext-FK über die *Zuständigkeitskette* weitergeleitet. Diese lose Kopplung ermöglicht eine unabhängige Entwicklung und eine höhere Wiederverwendbarkeit des Sub-Werkzeugs.

Da fachliche Änderungen nur über die Funktionskomponenten durchgeführt und an ihre Beobachter signalisiert werden, benötigt man keine Rückkopplung zwischen Sub-IAKs und Kontext-IAK. Die Benutzt-Beziehung zwischen Kontext-IAK und

Sub-IAKs wird nur aus technischen Gründen benötigt, z.B. für die Erzeugung und das Löschen von Sub-Werkzeugen.

Das Kontext-Werkzeug (siehe Abbildung 7), der *Werkzeugabschluss*, ist zuständig für das Erzeugen der Kontext-FK und der Kontext-IAK. Es ist ein Werkzeug-Objekt, das ein Werkzeug als Ganzes repräsentiert und entsprechend notwendige Informationen über das Werkzeug verwaltet.

3.3 Werkzeugkonstruktion in JWAM 1.4

Die Klassen, die den Werkzeugentwurf in JWAM umsetzen, sind in den Sub-Rahmenwerken der Handhabungs- und Präsentationsschicht (siehe Abbildung 4) „Tool Construction“, „GUI“ und „Environment“ implementiert. Hier werden die nicht-fachlichen, generischen Teile der Werkzeuge, wie das Erzeugen und Löschen, Anbindung an ein GUI-System und Verknüpfung mit Materialien, implementiert, so dass sich der Entwickler auf die fachliche Funktionalität der Werkzeuge konzentrieren kann.

Zentral für die Werkzeugimplementation mit JWAM sind die Klassen *toolObject*, *fpObject* und *ipObject*, die Teile des „Tool Construction“ Sub-Rahmenwerks sind. Diesen Zusammenhang haben wir in Abbildung 8 verdeutlicht. Der graue Kasten stellt das Sub-Rahmenwerk „Tool Construction“ mit dort angesiedelten Schnittstellen und den oben erwähnten Klassen dar. Die konkreten Klassen (außerhalb des Kastens), aus denen ein Werkzeug besteht, erben jeweils von den Klassen des Rahmenwerks und erweitern diese.

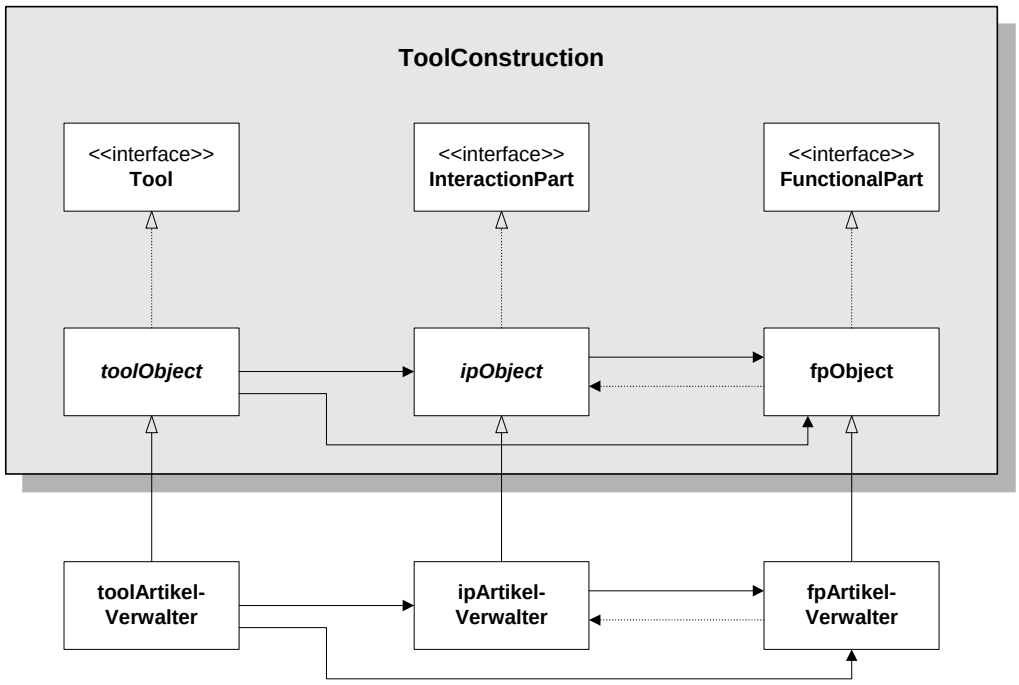

Abbildung 8: Klassen eines Werkzeugs

toolObject implementiert die generischen Operationen eines Werkzeugs, die in der *Tool*-Schnittstelle definiert sind. Die Klasse *toolObject* bildet die Basis für jedes Werkzeug. Die Basisklasse *toolObject* implementiert Operationen zum Erzeugen

der Funktionskomponente (`createFP()`) und der Interaktionskomponente (`createIP(fpObject fp)`), sowie das Starten und Schließen des Werkzeugs und die Verknüpfung mit Materialien. Die Methoden der *Tool*-Schnittstelle haben wir nachfolgend dargestellt.

```
public interface Tool extends Thing
{
  public boolean canStartWithoutMaterial ();
  public void start ();
  public void start (Thing material);
  public void show ();
  public void close ();
  public void setMaterial (Thing material);
  public boolean hasAspectFor (Thing material);
  public RequestController requestController ();
  public void setRequestSuccessor (RequestController successor);
  public FunctionalPart functionalPart ();
  public InteractionPart interactionPart ();
}
```

Die Funktionskomponente wird durch die Klasse *fpObject* realisiert. Diese Klasse implementiert die Operationen der *FunctionalPart*-Schnittstelle. Eine Funktionskomponente besitzt keine Informationen über ihre Interaktionskomponente, daher kann sie von unterschiedlichen Interaktionskomponenten benutzt werden. In der *fpObject*-Klasse ist der generelle Umgang mit Materialien, wie das Setzen oder Aktualisieren eines gegebenen Materials, bereits umgesetzt. Das Erzeugen von komplexen Kontext-FKs, die, je nach den zu bearbeitenden Aufgaben, aus mehreren Sub-FKs bestehen können, ist ebenfalls möglich. Das Erzeugen und Schließen der Sub-FKs muss in der konkreten FK-Klasse realisiert werden, da nur diese das dafür notwendige fachliche Wissen besitzt. Die Kommunikation zwischen der Funktionskomponente und ihren Sub-FKs erfolgt über Ereignisse (*events*) und Anfragen (*requests*).

Da die Interaktionskomponente auf der Funktionskomponente arbeitet, enthält die Oberklasse für alle Interaktionskomponenten, *ipObject*, einen Verweis auf *fpObject.* Bei der Erzeugung einer Interaktionskomponente muss die Funktionskomponente übergeben werden. Die Interaktionskomponente kann sich für bestimmte Ereignisse ihrer Funktionskomponente registrieren. Die Interaktionskomponente wird an die Benutzungsoberfläche (*GUIContext*) angebunden. Äquivalent zu der Funktionskomponente kann auch hier eine Kontext-Interaktionskomponente mit einer oder mehreren Sub-IAKs erzeugt werden.

Der GUI-Kontext bildet eine Abstraktion über die grundlegenden Mechanismen zur Präsentation an der Benutzungsoberfläche, sowie zur Interaktion des Benutzers mit dem Werkzeug. Der GUI-Kontext ermöglicht die Anbindung von Benutzungsoberflächen, die auf unterschiedlichen GUI-Technologien basieren, ohne dass die entwickelten Anwendungen geändert werden müssen. Die Benutzungsoberfläche kann beispielsweise auf AWT oder auf Swing basieren, wobei die Oberflächen beliebig ausgetauscht werden können.

3.4 Begleitendes Beispiel

Um das Konzept zu verdeutlichen, führen wir ein Beispielwerkzeug ein. Dieses Werkzeug, der Artikelverwalter, soll für das Verwalten von Zeitschriftenartikeln zuständig sein.

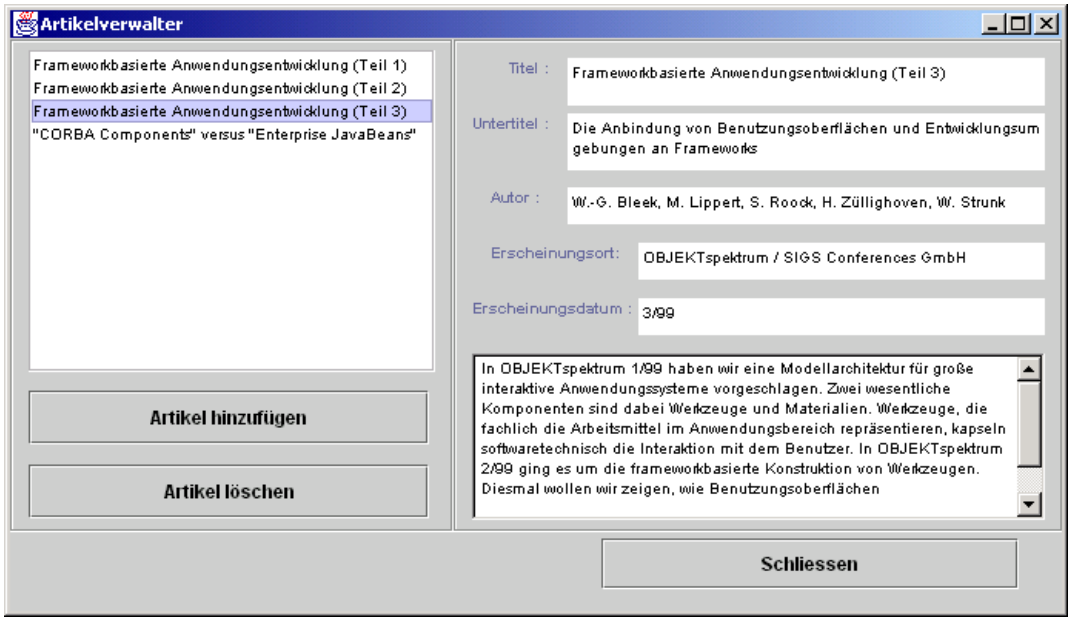

Abbildung 9: Artikelverwalter-Werkzeug

Das Artikelverwalter-Werkzeug kann in ein Kontext-Werkzeug und zwei Sub-Werkzeuge aufgeteilt werden. Ein Sub-Werkzeug Artikelauflister bietet die Möglichkeit zum Auflisten aller vorhandenen Artikel sowie zur Selektion eines Artikels. Mit Hilfe des Artikelbearbeiter-Werkzeugs können Informationen zu einem selektierten Artikel bearbeitet oder Informationen zu einem neuen Artikel erfasst werden.

Das Kontext-Werkzeug koordiniert die Zusammenarbeit der beiden Sub-Werkzeuge. Die gesamte Benutzungsoberfläche, wie in Abbildung 9 dargestellt, wird von dem Kontext-Werkzeug angeboten, so dass diese von den Sub-Werkzeugen mitbenutzt werden kann.

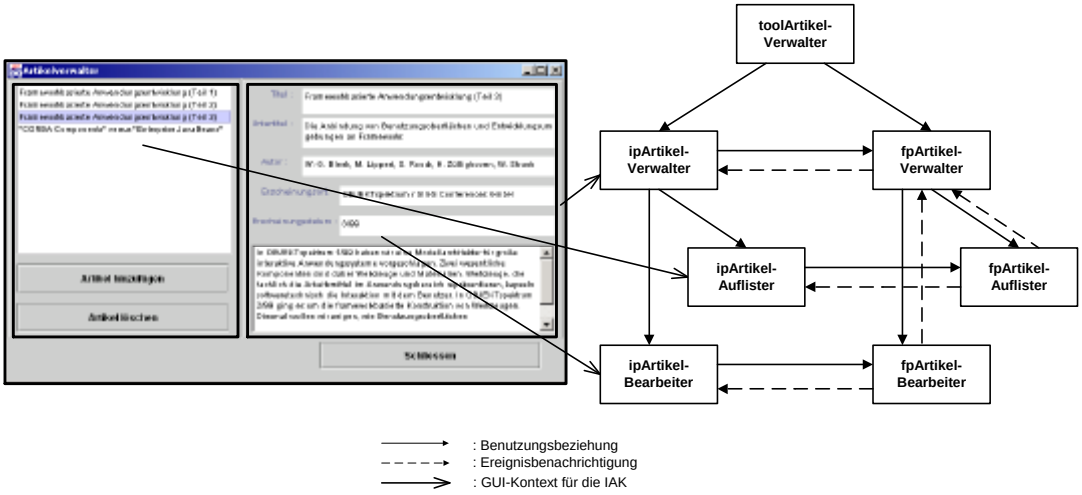

Abbildung 10: Artikelverwalter-Werkzeug nach WAM

In Abbildung 10 haben wir das Verhältnis der Interaktionskomponenten zu der Benutzungsoberfläche dargestellt. Die einzelnen Bereiche der Benutzungsoberfläche sind mit einem Rahmen versehen. Die Pfeile zeigen, von welcher Interaktionskomponente der jeweilige Bereich genutzt wird.

In der Benutzungsoberfläche des Artikelverwalter-Werkzeugs wird nur die „Schliessen“-Schaltfläche von dem Kontext-Werkzeug benutzt. Allerdings gehört die Fläche, auf der die übrigen graphischen Elemente platziert sind, zu dem Kontext-Werkzeug, daher ist die gesamte Benutzungsoberfläche umrandet und *ipArtikelVerwalter* zugeordnet.

Die Liste der vorhandenen Artikel, sowie die beiden Schaltflächen, „Artikel hinzufügen“ und „Artikel löschen“, repräsentieren das Artikelauflister-Sub-Werkzeug. Die übrigen grafischen Elemente stellen das Artikelbearbeiter-Sub-Werkzeug dar.

Selektiert der Benutzer einen Artikel aus der Liste, so werden die vorhandenen Informationen in dem Artikelbearbeiter angezeigt. Durch Betätigen der „Artikel löschen“-Schaltfläche kann ein selektierter Artikel gelöscht werden. Mit Hilfe der „Artikel hinzufügen“-Schaltfläche wird ein neuer Artikel erzeugt, wobei die Liste um einen Artikel mit dem Standardtitel „neuer Artikel“ erweitert wird. Dieser Artikel wird selektiert dargestellt und die Felder des Artikelbearbeiters erscheinen leer bis auf den Titel, der überschrieben werden muss. Weitere Informationen können nun in dem Artikelbearbeiter erfasst werden.

3.5 Technische Umsetzung

Technisch wird das Werkzeug durch das Werkzeug-Objekt *toolArtikelVerwalter* repräsentiert. Die Klasse *toolArtikelVerwalter* erbt von der abstrakten Klasse *toolObject* und muss dabei folgende Methoden überschreiben:

```
protected fpObject createFP ()
{
   fpObject fp = new fpArtikelVerwalter(requestController());
   Contract.ensure(fp != null, this);
   return fp;
}

protected ipObject createIP (fpObject fp)
{
   GUIManager guiMan = Environment.instance().guiManager();
   guiMan.register( new ClassResource("ArtikelVerwalter",
        guiArtikelVerwalter.class ) );

   if (fp instanceof fpArtikelVerwalter)
   {
    ipObject ip = new ipArtikelVerwalter( new guiArtikelVerwalter(),
       (fpArtikelVerwalter) fp );
   }

   Contract.ensure(ip != null, this);
   return ip;
}
```

In der Methode `createFP()` wird die Kontext-Funktionskomponente, ein Exemplar der Klasse *fpArtikelVerwalter*, erzeugt, und analog in der Methode `createIP (fpObject fp)` ein Exemplar der Klasse *ipArtikelVerwalter*.

Das Erzeugen der Subwerkzeuge haben wir mit Hilfe eines Interaktionsdiagramms in der Abbildung 11 verdeutlicht.

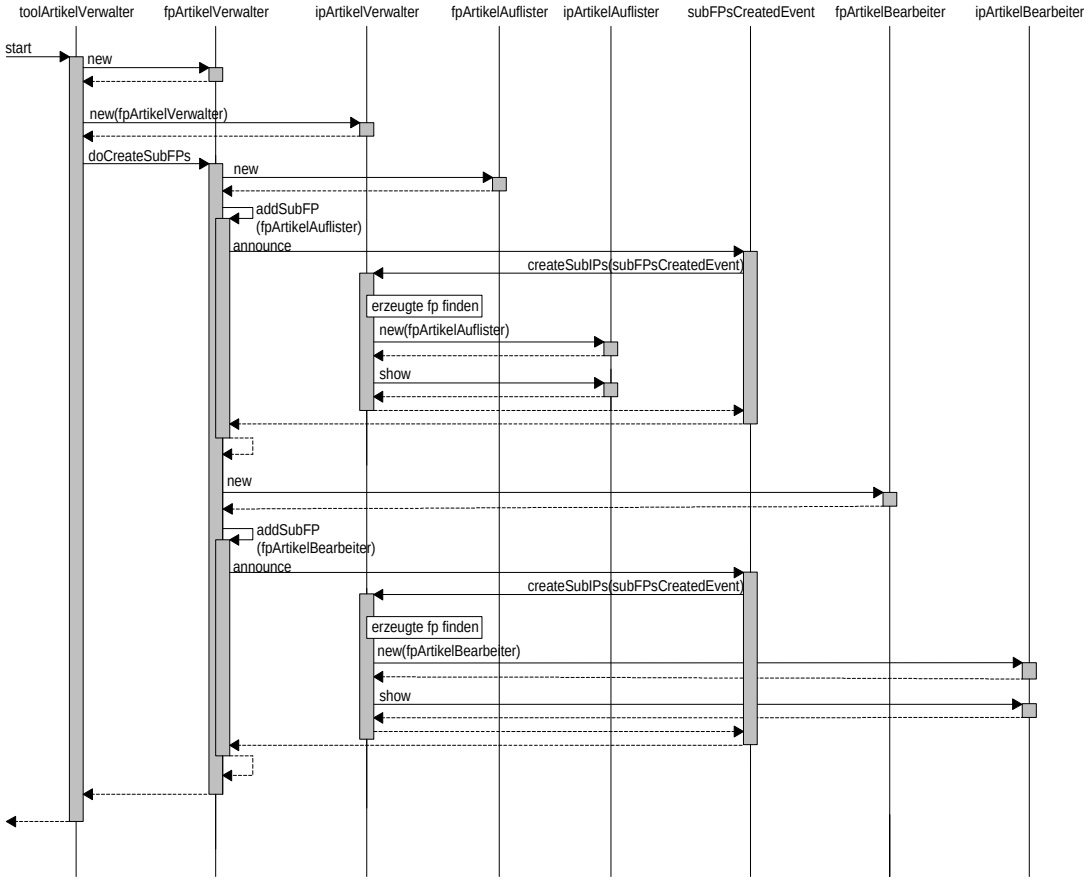

Abbildung 11: Interaktionsdiagramm –Erzeugung der Subwerkzeuge

Die Kontext-Funktionskomponente, *fpArtikelVerwalter*, ist für das Erzeugen der Sub-Funktionskomponenten zuständig. Dazu muss der Entwickler die Methode `doCreateSubFPs()` überschreiben. In dieser Methode wird festgelegt, welche Sub-Funktionskomponenten benötigt werden. In unserem Beispiel wird zuerst die *fpArtikelAuflister*-FK erzeugt und über den Aufruf von `addSubFP(fpArtikelAuflister)` an die Kontext-Funktionskomponente gebunden. Darüber hinaus wird ein Ereignis, `subFPsCreatedEvent`, ausgelöst und damit die Erzeugung der Sub-Interaktionskomponenten angestoßen. Zu jeder Funktionskomponente wird eine Interaktionskomponente erzeugt und an die Kontext-Interaktionskomponente angebunden. In diesem Fall wird die Methode `doNewSubIP` überschrieben, so dass die entsprechende Sub-Interaktionskomponenten erzeugt werden können. Der nachfolgende Quellcode zeigt eine Möglichkeit für die Erzeugung der Sub-Interaktionskomponenten. Dieser Quellcode stammt aus der Klasse *ipArtikelVerwalter*.

```
protected ipObject doNewSubIP (fpObject subFP)
{
   ipObject result = null;
   if (subFP instanceof fpArtikelAuflister)
   {
      result = new ipArtikelAuflister((IAFContext)context(),
                     (fpArtikelAuflister) subFP);
   };
```

```
    if (subFP instanceof fpArtikelBearbeiter)
    {
       result = new ipArtikelBearbeiter((IAFContext)context(),
                                (fpArtikelBearbeiter) subFP);
    };
    return result;
 }
```

Die Sub-Funktionskomponente *fpAuflisterBearbeiter* sowie ihre Interaktionskomponente werden auf die bereits beschriebene Weise erzeugt.

3.6 Kritik an der Werkzeugkonstruktion mit JWAM 1.4

Bei der Weiterentwicklung des Rahmenwerks kommt es häufig vor, dass man eine neu eingebaute oder überarbeitete Funktionalität mit einem Werkzeug testen muss. Der Entwickler baut zu diesem Zweck einen Prototypen. Die Anforderungen an einen Prototypen sind andere als an „fertige“ Werkzeuge. Es kommt bei den Prototyp nicht auf die Vollständigkeit an, sondern der Entwickler konzentriert sich auf eine bestimmte Funktionalität, die er untersuchen will. Der Zeitaufwand für das Erstellen des Prototypen sollte so gering wie möglich sein, damit der Entwickler sich auf die inhaltliche Aufgabe konzentrieren kann. In der Version 1.4 des JWAM-Rahmenwerks gibt es keine gesonderte Unterstützung für die Erstellung eines solchen Werkzeugs. Selbst die Erzeugung von Werkzeugen ohne Funktionalität ist mit viel Tipparbeit verbunden. Es müssen mindestens drei Klassen erzeugt werden. Eine Klasse für die Funktionskomponente, eine Klasse für die Interaktionskomponente und eine Klasse für das Werkzeug. Wobei noch hinzukommt, das man bestimmte Methoden überschreiben muss.

Mit zunehmender Größe des Rahmenwerks nimmt auch die Zahl der bereits entwickelten Werkzeuge und Subwerkzeuge zu. Die Frage ob, und wo eine bestimmte Funktionalität schon implementiert wurde, wird mit zunehmender Komplexität immer schwieriger zu beantworten sein. In der JWAM-Version 1.4 gibt es keinen Mechanismus, der den Entwickler bei der Wiederverwendung unterstützt. Es kann sich für den Entwickler recht aufwendig gestalten, wenn er eine bestimmte Funktionskomponente wiederverwenden möchte. Falls die Funktionskomponente Sub-Funktionskomponenten besitzt, muss der Entwickler alle Sub-Funktionskomponenten auffinden.

Die Unterscheidung von Kontext-Werkzeug und Sub-Werkzeug auf der Modellierungsebene, wird in der JWAM-Version 1.4 auch auf die technische Realisierung übertragen. Die Konstruktion von Kontext-Werkzeugen unterscheidet sich von der Konstruktion der Sub-Werkzeuge. Kontex-Werkzeuge besitzen eine Werkzeughülle, das Werkzeug-Objekt. Sub-Werkzeuge besitzen dagegen kein Werkzeug-Objekt. Wir sind der Meinung, dass auf der technischen Realisierungsseite diese Trennung nicht nötig ist. Es ist einfacher und flexibler, wenn man technisch eine Art von Werkzeug hat. Diese kann man entweder als Kontext- oder Sub-Werkzeug einsetzen.

3.7 Zusammenfassung

In diesem Kapitel haben wir die Werkzeugkonstruktion nach dem WAM-Ansatz sowie das JWAM-Rahmenwerk vorgestellt. Darüber hinaus haben wir erläutert, auf welche Weise das JWAM-Rahmenwerk die Entwicklung von Werkzeugen unterstützt.

Das von uns eingeführte Beispiel-Werkzeug wird in den nachfolgenden Kapiteln verwendet, um das vorgestellte Konzept und seine Umsetzung zu verdeutlichen.

Ein Kombi-Werkzeug besteht aus mehreren Sub-Werkzeugen. Jedes dieser Sub-Werkzeuge ist für die Bearbeitung bestimmter in sich abgeschlossener Teilaufgaben zuständig. Die Sub-Werkzeuge kennen weder sich untereinander, noch das Kontext-Werkzeug. Daher erfolgt die Rückkopplung von einem Sub-Werkzeug zu seinem Kontext-Werkzeug über den Ereignismechanismus. Das Kontext-Werkzeug kennt dagegen seine Sub-Werkzeuge und nimmt ihre Dienste über deren fachliche Schnittstelle (die Schnittstelle der Funktionskomponente) in Anspruch. Das Kombi-Werkzeug wird durch ein Werkzeug-Objekt repräsentiert.

Die Kritik an der Werkzeugkonstruktion mit der JWAM-Version 1.4 ist keine Fundamentalkritik, sondern es sind Verbesserungsmöglichkeiten, die eine effizientere und flexiblere Werkzeugentwicklung zur Folge hätten. Die einzelnen Kritikpunkte fassen wir zusammen:

- Es gibt in der Version 1.4 des JWAM-Rahmenwerks keine einfache Möglichkeit für Testzwecke Prototypen zu erstellen.
- Es gibt keinen Mechanismus der den Entwickler bei der Wiederverwendung unterstützt.
- Die Konstruktion der Kontext-Werkzeuge unterscheidet sich von der Konstruktion der Sub-Werkzeuge.

4 Werkzeugkomponenten

Die Kritik an der Werkzeugkonstruktion mit der Version 1.4 des JWAM-Rahmenwerks bildet unsere Motivation für den Einsatz von Komponenten. Wir erhoffen uns dadurch eine Verbesserung der Wiederverwendbarkeit von einzelnen Werkzeugen oder auch Werkzeugteilen, sowie eine leichtere Einarbeitung in die Werkzeugkonstruktion mit dem JWAM-Rahmenwerk.

In diesem Kapitel stellen wir unser Konzept einer Werkzeugentwicklung mit Komponenten vor.

4.1 Konzept der Werkzeugkomponente

Das in Kapitel 3 vorgestellte Werkzeug-Konzept des WAM-Ansatzes kann weitgehend auf Komponenten übertragen werden. In der komponentenorientierten Sicht wird ein Werkzeug als eine Werkzeugkomponente realisiert.

Eine Werkzeugkomponente realisiert eine bestimmte fachliche Funktionalität und regelt die Interaktion mit dem Benutzer. Sie ermöglicht das Verändern und Sondieren von Materialien. Nach außen enthüllt eine Werkzeugkomponente keine Details über ihren Aufbau. Die Trennung zwischen Funktionskomponente und Interaktionskomponente kann innerhalb der Werkzeugkomponente vorliegen, ist aber nicht zwingend notwendig. Die fachliche Funktionalität der Werkzeugkomponente kann von anderen Werkzeugkomponenten über eine wohldefinierte Schnittstelle benutzt werden. Nach Fröse (vgl. [Fröse99]) kann diese Schnittstelle als Call-In-Schnittstelle bezeichnet werden. Im Unterschied zu dem im Kapitel 3 erläuterten Werkzeug-Konzept wird jede Werkzeugkomponente zur Entwurfszeit durch eine Werkzeug-Klasse und zur Laufzeit durch ein Werkzeug-Objekt, das Exemplar der Werkzeug-Klasse, nach außen repräsentiert. Über das Werkzeug-Objekt macht eine Werkzeugkomponente ihre fachliche Schnittstelle bekannt, und fordert Dienste anderer Werkzeugkomponenten, in Form einer ihr bekannten fachlichen Schnittstelle, an.

Werkzeugkomponenten können hierarchisch zu komplexen Werkzeugkomponenten zusammengesetzt werden. Innerhalb der Hierarchie nehmen die Werkzeugkomponenten unterschiedliche Rollen ein. Eine zusammengesetzte Werkzeugkomponente, die aus anderen Werkzeugkomponenten besteht, nennen wir im folgenden Kontext-Werkzeugkomponente. Die jeweiligen Sub-Werkzeuge werden als Sub-Werkzeugkomponenten bezeichnet.

In Abbildung 12 haben wir die zusammengesetzte Artikelverwalter-Werkzeugkomponente dargestellt. Das in Kapitel 3 von uns eingeführte Beispielwerkzeug kann in drei Werkzeugkomponenten realisiert werden: Artikelverwalter-, Artikelauflister- und Artikelbearbeiter-Werkzeugkomponente. Die innere Struktur der einzelnen Werkzeugkomponenten, d.h. die Trennung in eine Funktionskomponente und eine Interaktionskomponente ist nicht mehr sichtbar.

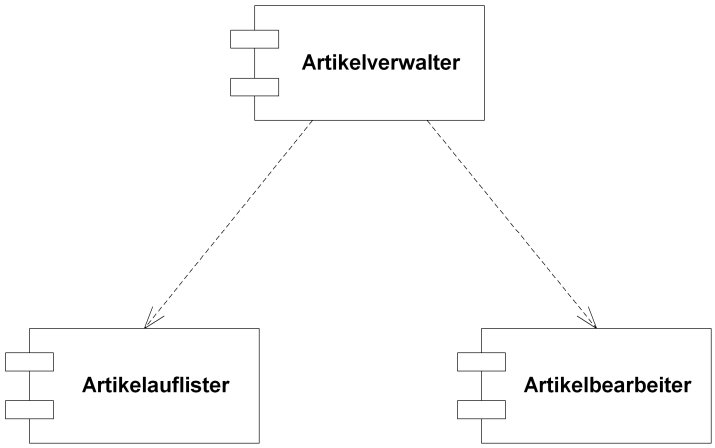

Abbildung 12: Werkzeugkomponenten des Artikelverwalters

Eine Kontext-Werkzeugkomponente übernimmt die Koordination der Aufgaben. Sie beauftragt entsprechend ihre Sub-Werkzeugkomponenten, bestimmte Teilaufgaben zu bearbeiten. Anlehnend an das Bürokratie-Muster (vgl. [Riehle98]) spricht Fröse von einer Vorgesetztenkomponente und ihren Untergebenen. Fröse wendet das Bürokratie-Muster für den komponentenorientierten Entwurf von Werkzeugen an, und beschreibt die Eigenschaften einer Werkzeugkomponente, die sich daraus ergeben (vgl. [Fröse99], s.31).

4.2 Modellarchitektur

Im folgenden stellen wir unsere Modellarchitektur für komponentenorientierte Werkzeugarchitekturen vor. Dabei beschreiben wir die Art der verwendeten Entwurfselemente sowie deren Verbindungen.

Züllighoven (vgl. [Züllighoven98]) definiert den Begriff der Modellarchitektur:

> *„Eine Modellarchitektur beschreibt die allgemeinen Prinzipien hinter einer Softwarearchitektur. Sie umfasst die grundlegenden Elemente, deren Verknüpfungen und die Regeln, die für eine Softwarearchitektur gelten.*
>
> *Eine Modellarchitektur gibt Anleitung bei der softwaretechnischen Realisierung eines Softwaresystems.“*

Die Elemente der hier vorgestellten Modellarchitektur sind einfache sowie zusammengesetzte Werkzeugkomponenten, ein Werkzeugkomponentenmanager und die Verbindungsstücke Operationsaufrufe, Ereignisse (*events*) und Anfragen (*requests*), sowie die Regeln.

4.2.1 Werkzeugkomponente

Für uns stellt eine Werkzeugkomponente eine nach außen abgeschlossene Modellierungseinheit dar. Jede Werkzeugkomponente implementiert eine allgemeine Werkzeugschnittstelle (*Tool*). Die Werkzeugschnittstelle definiert alle generischen Operationen einer Werkzeugkomponente (genaue Beschreibung folgt in Kapitel 5). Dadurch, dass jede Werkzeugkomponente diese Schnittstelle implementiert, wird der Umgang mit Werkzeugkomponenten vereinheitlicht.

Nach außen wird jede Werkzeugkomponente durch ein Werkzeug-Objekt repräsentiert. Das Werkzeug-Objekt ist wiederum ein Exemplar der Werkzeugkomponentenklasse. Konkrete Werkzeugkomponenten werden durch Vererbung gebildet.

Die genaue Beschreibung einer Werkzeugkomponente kann über ihre Schnittstelle angefordert werden. Dazu gehört der Name der Werkzeugkomponente, die fachliche Schnittstelle (Call-In-Schnittstelle) sowie die Schnittstellen, die benötigt werden, um alle Dienstleistungen erfüllen zu können (Call-Out-Schnittstellen, vgl. [Fröse99]). Darüber hinaus muss eine Werkzeugkomponente Auskunft geben, mit welcher Art von Materialien (Aspekten) sie umgehen kann.

Wie oben erläutert, erfüllt eine Werkzeugkomponente bestimmte fachliche Funktionalität, die zur Bearbeitung einer Aufgabe dient. Jede Werkzeugkomponente bietet ihre Funktionalität in Form einer Schnittstelle anderen Werkzeugkomponenten an. Konkrete fachliche Funktionalität wird durch Vererbung gebildet. Die Werkzeugkomponentenfunktionalität spezifiziert die *ToolFunctionality*-Schnittstelle, die den allgemeinen Umgang mit der Funktionalität eines Werkzeugs beschreibt.

Eine Werkzeugkomponente realisiert außerdem ihre Interaktion. Die Interaktion beinhaltet die Handhabung einer Werkzeugkomponente. Sie sorgt für die Anbindung der Benutzungsoberfläche und die Interaktion mit dem Benutzer. Die Präsentation einer Werkzeugkomponente wird durch die Benutzungsoberfläche (View) umgesetzt. Dagegen ist *ToolInteraction* eine abstrakte Klasse, die eine potentielle Konstruktion von Interaktion beschreibt. Eine Komponente kann diese abstrakte Klasse erweitern und dadurch ihre Interaktion definieren. Eine Sub-Werkzeugkomponente verwendet entweder ihre eigene Benutzungsoberfläche, indem sie diese in die Benutzungsoberfläche der Kontext-Werkzeugkomponente einbettet, oder sie nutzt die Oberfläche der Kontext-Werkzeugkomponente, die die gesamte Darstellung zur Verfügung stellt. Eine Werkzeugkomponente kann auch unterschiedliche Benutzungsoberflächen anbieten, die beispielsweise abhängig von einem bestimmten GUI-Rahmenwerk sein können. Denkbar wäre beispielsweise eine Realisierung mit Swing oder AWT. Die Interaktion der Werkzeugkomponente bleibt nach außen verborgen. Über die Schnittstelle einer Werkzeugkomponente kann ihre Benutzungsoberfläche (View), aber nicht ihre Interaktion (*ToolInteraction*) angefordert werden.

Man unterscheidet einfache und zusammengesetzte Werkzeugkomponenten. Äquivalent zu dem im Kapitel 3 vorgestellten Werkzeug-Konzept, beinhaltet eine einfache Werkzeugkomponente keine Sub-Werkzeugkomponenten. Eine solche Werkzeugkomponente erfüllt ihre fachliche Aufgabe eigenständig, ohne den Zugriff auf die Funktionalität anderer Werkzeugkomponenten. Ein Beispiel für eine einfache Werkzeugkomponente ist die Auflister-Komponente, die wir in Abbildung 13 dargestellt haben. Die Auflister-Komponente bietet über ihre Tool-Schnittstelle den Zugriff auf ihre fachliche Funktionalität und die Benutzungsoberfläche View.

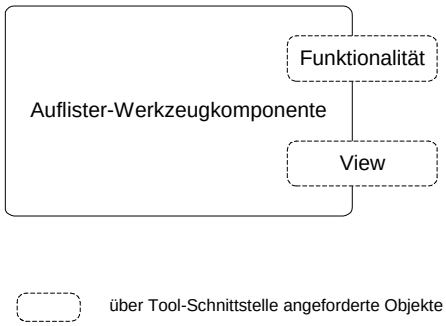

Abbildung 13: Einfache Auflister-Werkzeugkomponente

Bei komplexen Werkzeugen ist eine Aufteilung der Funktionalität auf mehrere Werkzeugkomponenten sinnvoll. Eine zusammengesetzte Werkzeugkomponente stellt ein Kombi-Werkzeug dar. Softwaretechnisch besteht ein Kombi-Werkzeug aus einer Kontext-Werkzeugkomponente und mehreren Sub-Werkzeugkomponenten. Jede dieser Sub-Werkzeugkomponenten übernimmt konkrete fachliche Aufgaben, und vervollständigt damit die Gesamtfunktionalität des Werkzeugs.

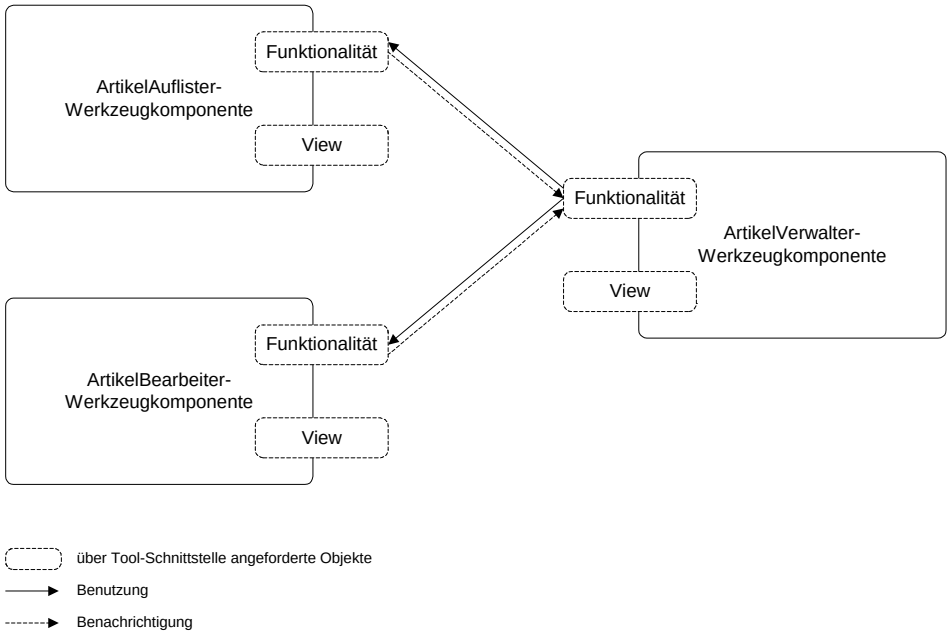

Abbildung 14: Zusammengesetzte Artikelverwalter-Werkzeugkomponente

In Abbildung 14 haben wir eine zusammengesetzte Artikelverwalter-Werkzeugkomponente dargestellt. Sie besteht aus drei Teilen: der Kontext-Werkzeugkomponente „ArtikelVerwalter“ und den beiden Sub-Werkzeugkomponenten „ArtikelAuflister“ und „ArtikelBearbeiter“. Alle dargestellten Werkzeugkomponenten bieten die Möglichkeit, über die Tool-Schnittstelle ihre Funktionalität anzufordern. Neben dieser fachlichen Schnittstelle stellen die Werkzeugkomponenten eine View-Schnittstelle zur Verfügung. Diese kann ebenfalls nur über die Tool-Schnittstelle angefordert werden. Die

durchgezogenen Pfeile zeigen die Richtung der Benutzt-Beziehung. Im Gegenteil dazu stellen die gestrichelten Pfeile die Ereignisbenachrichtigung dar. Aus der Abbildung wird deutlich, dass die Funktionalität der Kontext-Werkzeugkomponente um die Funktionalität der Sub-Werkzeugkomponenten erweitert wird. Eine zusammengesetzte Werkzeugkomponente kann rekursiv einer anderen Werkzeugkomponente untergeordnet werden. Sie erfüllt dann für ihre Sub-Werkzeugkomponenten die Rolle eines Vorgesetzten, gleichzeitig fungiert sie als eine Untergeordnete ihrer Kontext-Werkzeugkomponente.

Zusammenfassend halten wir als Ergebnisse fest:

> Konkrete Werkzeugkomponenten werden durch Vererbung gebildet. Jede Werkzeugkomponente implementiert die Tool-Schnittstelle, was den Umgang mit Werkzeugkomponenten vereinheitlicht. Jede Werkzeugkomponente bietet eine fachliche Funktionalität an. Konkrete fachliche Funktionalitäten werden durch Vererbung gebildet.

4.2.2 Werkzeugkomponentenmanager

Die fachliche Funktionalität einer Werkzeugkomponente wird softwaretechnisch durch die Spezialisierung der allgemeinen Funktionalität-Schnittstelle realisiert. Dadurch erhält die Funktionalität einen Typ. Der Typ der Funktionalität, die eine Werkzeugkomponente als ihren Dienst anbietet, kann über ihre Schnittstelle abgefragt werden.

Für die Anbindung einer Sub-Werkzeugkomponente ist deren fachliche Funktionalität entscheidend. Stimmt der Typ der fachlichen Funktionalität mit dem Typ überein, die eine Kontext-Werkzeugkomponente erwartet, so kann die Sub-Werkzeugkomponente eingebunden werden. Es besteht die Möglichkeit, dass mehrere Werkzeugkomponenten die Funktionalität gleichen Typs realisieren, sich aber beispielsweise in der Darstellung unterscheiden. So kann eine Artikelauflister-Werkzeugkomponente alle vorhandenen Artikel in einer einfachen Liste darstellen, ein anderer Artikelauflister dagegen in einer Tabelle. In den meisten Fällen spielt es keine Rolle, welche Werkzeugkomponente die geforderte Funktionalität erfüllt.

Zu den Aufgaben des Werkzeugkomponentenmanagers gehört die Verwaltung aller vorhandenen Werkzeugkomponenten. Er übernimmt keine koordinierende Rolle bei der Zusammenarbeit der einzelnen Werkzeugkomponenten, sondern sorgt für die Erstellung und Verknüpfung von Werkzeugkomponenten. Um dies zu gewährleisten meldet sich jede Werkzeugkomponente beim Werkzeugkomponentenmanager an. Diesen Zusammenhang haben wir in Abbildung 15 verdeutlicht. Der Manager ist nach dem Singletonmuster (vgl. [Gamma96]) realisiert.

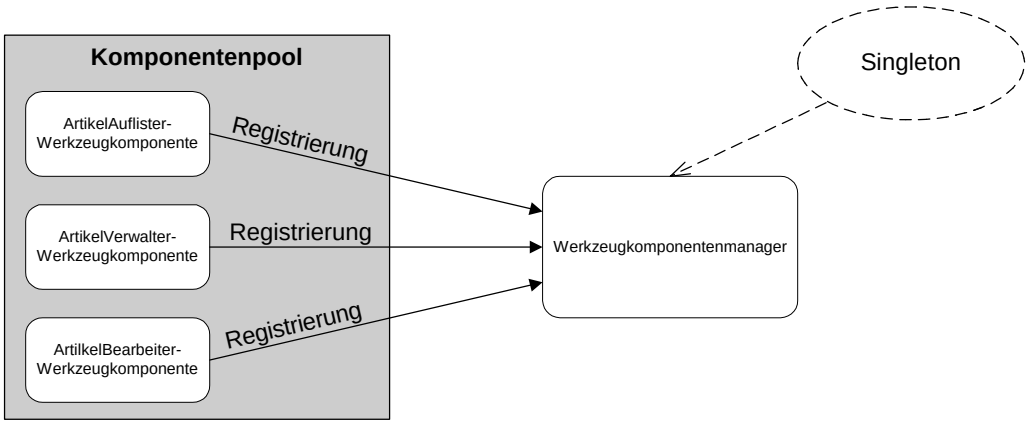

Abbildung 15: Werkzeugkomponentenmanager

Während der Laufzeit wendet sich die Kontext-Werkzeugkomponente an den Werkzeugkomponentenmanager mit der Anforderung einer bestimmten Funktionalität. Der Werkzeugkomponentenmanager ist dafür zuständig, aus den bei ihm angemeldeten Werkzeugkomponenten die passende zu finden und sie zu erzeugen. Danach übergibt der Manager die neuerzeugte Werkzeugkomponente an die anfragende Kontext-Werkzeugkomponente. Diese sorgt für die Anbindung der neuen Sub-Werkzeugkomponente und erweitert ihre fachliche Funktionalität um die Dienste der Sub-Werkzeugkomponente. Die Kontext-Werkzeugkomponente delegiert Aufgaben an ihre Sub-Werkzeugkomponenten. Die untergeordneten Werkzeugkomponenten informieren ihre übergeordnete Kontext-Werkzeugkomponente über ihre Zustandsänderungen mit Hilfe des Ereignismechanismus.

Wir halten folgendes Ergebnis fest:

> Der Werkzeugkomponentenmanager ermöglicht flexible Konstruktion von Werkzeugkomponenten.

4.2.3 Verbindungsstücke

Bäumer (vgl. [Bäumer98]) definiert den Begriff Verbindungsstück bezogen auf objektorientierte Softwarearchitekturen wie folgt:

„Ein Verbindungsstück beschreibt, wie zwei oder mehr Komponenten miteinander gekoppelt werden und wie sie interagieren. Verbindungsstücke erfüllen somit primär einen softwaretechnischen Zweck. Im Kontext objektorientierter Softwareentwicklung lassen sich Entwurfmuster zur Beschreibung komplexer Verbindungsstücke verwenden.“

Werkzeugkomponenten werden über unterschiedliche Verbindungsstücke miteinander gekoppelt. Dazu gehören Operationsaufrufe, Ereignisse (*events*) sowie Anfragen (*requests*).

Mittels Operationsaufrufen kann eine Kontext-Werkzeugkomponente sondierend oder verändernd auf ihre Sub-Werkzeugkomponenten zugreifen. Entsprechende Zusicherungen werden in einem Vertrag (siehe [Meyer97]) festgelegt. Beide Seiten verpflichten sich, unter vereinbarten Bedingungen den Vertrag zu erfüllen. Bei einem Operationsaufruf fordert die Kontext-Werkzeugkomponente eine

Dienstleistung ihrer Sub-Werkzeugkomponente an. Die Kontext-Werkzeugkomponente muss die festgelegten Vorbedingungen[1] erfüllen. Die Kontext-Werkzeugkomponente muss sicherstellen, dass vor dem Aufruf der Operation die entsprechenden Bedingungen erfüllt sind. Damit die Überprüfung der Bedingungen erfolgen kann, muss die Sub-Werkzeugkomponente entsprechende Test-Operationen an ihrer fachlichen Schnittstelle zur Verfügung stellen. Erst wenn die Vorbedingungen erfüllt sind, kann der Operationsaufruf erfolgen. Die Aufgabe des Anbieters, der Sub-Werkzeugkomponente, besteht in der Ausführung der Operation und der Garantie der Korrektheit von Nachbedingungen.

Durch Operationsaufrufe kann die Kontext-Werkzeugkomponente Aufgaben an die Sub-Werkzeugkomponenten delegieren. Nachdem eine Operation ausgeführt worden ist, muss die Sub-Werkzeugkomponente ihre Kontext-Werkzeugkomponente entsprechend über relevante Zustandsänderungen informieren. Ereignisse ermöglichen die Rückkopplung von einer Sub-Werkzeugkomponente an ihre Kontext-Werkzeugkomponente. Die Kontext-Werkzeugkomponente kann sich für das Auftreten eines relevanten Ereignisses als Beobachter registrieren. Beim Auftreten des Ereignisses werden die Beobachter benachrichtigt, so dass sie entsprechend reagieren können. Beispielsweise kann der neue Zustand der Sub-Werkzeugkomponente oder des Materials abgefragt werden, um die Darstellung an der Benutzungsoberfläche zu aktualisieren. Für die Kontext-Werkzeugkomponente besteht keine Verpflichtung, sich für die angebotenen Ereignisse ihrer Sub-Werkzeugkomponenten zu registrieren oder auf Ereignisse zu reagieren.

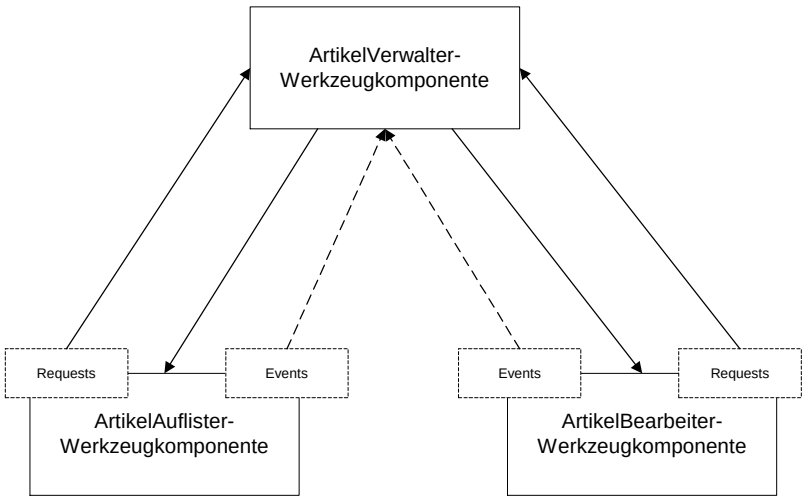

Abbildung 16: Kommunikation zwischen Werkzeugkomponenten

Innerhalb eines Werkzeugs sind die Sub-Werkzeugkomponenten ihrer Kontext-Werkzeugkomponente untergeordnet und haben kein Wissen über diese. Bestimmte Aufgaben, wie z.B. das Schließen eines Werkzeugs, kann eine Werkzeugkomponente nicht eigenständig erfüllen. Aus diesem Grund muss sie die Möglichkeit besitzen, ihre Aufträge entlang der Zuständigkeitskette zu delegieren. Jede Werkzeugkomponente ist in die Zuständigkeitskette integriert. Als

[1] Ein Vertrag bezieht sich immer auf einen Operationsaufruf.

RequestHandler ist eine Werkzeugkomponente verpflichtet, eine ihr übergebene Anfrage entweder abzuarbeiten oder an ihren Kontext weiterzuleiten. Die Zuständigkeitskette wird entsprechend dem Muster *Chain-of-Responsibility* (vgl. [Gamma96]) aufgebaut. Der oberste *RequestHandler* ist die Umgebung. Diese ist in der Lage unterschiedliche Anfragen zu behandeln, wie das Starten und Schließen einer Werkzeugkomponente oder auch das Speichern eines Materials.

4.2.4 Regeln

Die einzelnen Werkzeugkomponenten sollten lose gekoppelt sein und keine zyklische Strukturen aufweisen. Die Rückkopplung zwischen Werkzeugkomponenten sollten, wie oben beschrieben, durch Ereignisse und Anfragen gelöst werden.

Sub-Werkzeugkomponenten besitzen keine Kenntnis über die Kontext-Werkzeugkomponente und über deren Sub-Werkzeugkomponenten.

Jede Werkzeugkomponente hat innerhalb der Werkzeughierarchie einen *RequestHandler*, der ihre Anfragen abarbeitet oder entlang der Zuständigkeitskette weiterleitet.

4.3 Vergleich mit Werkzeugmodellarchitektur nach Fröse

Fröse unterscheidet zwischen Werkzeugkomponenten, zusammengesetzten Werkzeugkomponenten und Wurzelkomponenten. Jede dieser Werkzeugkomponenten muss mindestens die Schnittstellen Subordinate und View erfüllen. Mit der Bezeichnung Subordinate benennt Fröse gleichzeitig die Rolle, die eine Werkzeugkomponente unter dieser Schnittstelle einnimmt. Über die Subordinate-Schnittstelle stellt eine Werkzeugkomponente ihre Dienste der übergeordneten Werkzeugkomponente zur Verfügung. Fröse empfiehlt, für jeden Materialaspekt, mit dem eine Werkzeugkomponente arbeitet, eine eigene Schnittstelle bereitzustellen.

In unserer Modellarchitektur ermöglicht eine Werkzeugkomponente über die *Tool*-Schnittstelle den Zugriff auf ihre Dienste. Eine Werkzeugkomponente hat immer nur eine Funktionalität, die die *ToolFunctionality*-Schnittstelle implementiert. Wie wir im Abschnitt 4.2.1 erläutert haben, kann eine Werkzeugkomponente Auskunft darüber geben, mit welchen Aspekten sie arbeitet. Sie besitzt keine eigene Schnittstelle für jeden Aspekt. Ähnlich wie bei der Subordinate-Schnittstelle kann eine Werkzeugkomponente über *ToolFunctionality* in eine andere Werkzeugkomponente als Sub-Werkzeugkomponente eingebunden werden.

Im Unterschied zu den in unserer Modellarchitektur vorgestellten zusammengesetzten Werkzeugkomponenten, fordert Fröse, dass jede zusammengesetzte Werkzeugkomponente eine Manager-Schnittstelle implementieren muss. Die Manager-Schnittstelle ermöglicht einer übergeordneten Werkzeugkomponente, Anfragen (*requests*) zu empfangen. In unserem Modell implementiert jede Werkzeugkomponente die *RequestHandler*-Schnittstelle. Dadurch ist eine Werkzeugkomponente immer in der Lage, eine Anfrage zu empfangen und auf sie entsprechend zu reagieren, d.h. weiterzuleiten oder abzuarbeiten. Dieses gilt für alle Werkzeugkomponenten unabhängig davon, ob es sich um eine einfache oder eine zusammengesetzte Komponente handelt.

Für den Abschluss einer Werkzeugkomponentenhierarchie führt Fröse die Wurzelkomponente ein. Die Wurzelkomponente ist der oberste *RequestHandler* und übernimmt weitere Aufgaben wie die Darstellung unterschiedlicher graphischer Elemente (siehe [Fröse99] für eine genauere Beschreibung). Durch die Einbettung der Modellarchitektur in das JWAM-Rahmenwerk ist eine Wurzelkomponente als Abschluss nicht notwendig. Diese Aufgabe übernimmt die Umgebung.

Ein wesentlicher Unterschied zu der von Fröse vorgestellten Modellarchitektur für Werkzeugkomponenten ist der Werkzeugkomponentenmanager, der als zentralen Dienst die Verwaltung und Erstellung von Werkzeugkomponenten anbietet. Der Werkzeugkomponentenmanager gewährleistet eine effizientere und flexiblere Konstruktion von Werkzeugen.

4.4 Zusammenfassung

Werkzeugkomponenten bieten dem Entwickler eine effizientere und flexiblere Erstellung von Werkzeugen. Die unterschiedliche Repräsentation eines Kontext-Werkzeugs und eines Sub-Werkzeugs, wie sie in Kapitel 3 vorgestellt wurde, ist bei Werkzeugkomponenten nicht vorhanden. Jede Werkzeugkomponente wird nach außen durch ein Werkzeugobjekt repräsentiert. Das Konzept der Werkzeugkomponente und des Werkzeugkomponentenmanagers ermöglicht flexiblere Wiederverwendbarkeit von Werkzeugen als in der vorherigen Implementierung des Werkzeugkonzepts. Eine Sub-Werkzeugkomponente muss zwar den benötigten Typ der fachlichen Funktionalität erfüllen, den die Kontext-Werkzeugkomponente erwartet, kann aber ansonsten beliebig ausgetauscht werden.

5 Realisierung in JWAM

Im vorangegangenen Kapitel haben wir ein Konzept für den Entwurf von Werkzeugkomponenten beschrieben. In diesem Abschnitt erläutern wir die Realisierung des vorgestellten Konzeptes in der Sprache Java sowie die Integration in das JWAM-Rahmenwerk. Das von uns in Kapitel 4 vorgestellte Konzept ist in der Version 1.5 des JWAM-Rahmenwerks realisiert und wird zur Werkzeugkonstruktion eingesetzt.

5.1 Werkzeugkomponenten in JWAM

Innerhalb einer Anwendung sind Werkzeugkomponenten für die Veränderung und Sondierung von Materialien im Rahmen einer fachlichen Aufgabe zuständig. Die von jeder Werkzeugkomponente angebotenen Dienste sind in einer fachlichen Schnittstelle als eine Menge von Operationen definiert.

Das JWAM-Rahmenwerk gibt die Grundstruktur der Werkzeugkomponenten vor. Die generischen Teile einer Werkzeugkomponente können in dem Rahmenwerk bereits vorimplementiert werden. Dies sind insbesondere die technischen Teile der Werkzeugkomponenten. Der Vorteil, der sich aus der Verwendung eines Rahmenwerks ergibt, liegt darin, dass sich der Entwickler auf die Implementierung der fachlichen Funktionalität der Werkzeugkomponenten konzentrieren kann. Darüber hinaus können bereits implementierte Werkzeugkomponenten wiederverwendet werden.

Das JWAM-Rahmenwerk stellt unterschiedliche Dienste zur Verfügung. Das Erzeugen und Löschen von Werkzeugkomponenten, die Anbindung eines GUI-Systems, Kommunikation zwischen den Werkzeugkomponenten, sowie die Verknüpfung mit Materialien werden generisch in dem Rahmenwerk definiert.

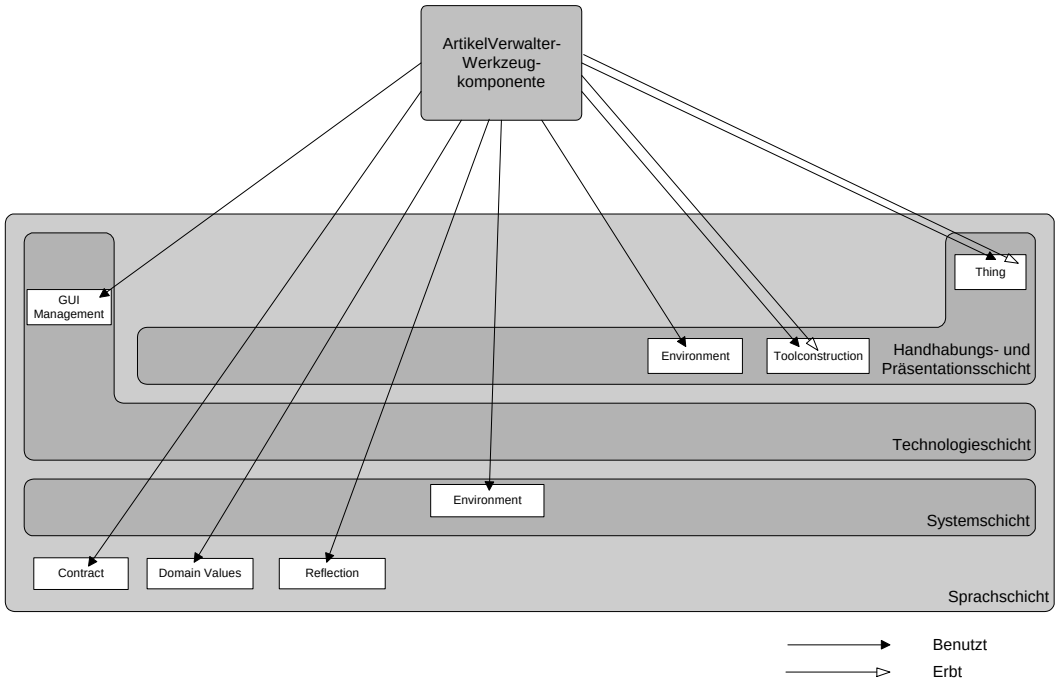

Abbildung 17: ArtikelVerwalter-Werkzeugkomponente und JWAM

Abbildung 17 stellt den Zusammenhang zwischen einer Werkzeugkomponente, dem Artikelverwalter, und dem JWAM-Rahmenwerk dar. Die Artikelverwalter-Werkzeugkomponente verwendet das JWAM-Rahmenwerk auf zwei unterschiedliche Arten. Auf der einen Seite erbt die Komponente von vorhandenen Klassen, wie der Vererbungspfeil in der Abbildung von der ArtikelVerwalter-Werkzeugkomponente zum „ToolConstruction“-Package aus der Handhabungs- und Präsentationsschicht zeigt. In diesem Package sind alle Klassen angesiedelt, die den Werkzeugbau unterstützen. Auf der anderen Seite werden vorhandene Klassen benutzt, wie beispielsweise der Benutztpfeil von der ArtikelVerwalter-Werkzeugkomponente zum Package „Contract“ aus der Sprachschicht zeigt. In dem Package „Contract“ sind Klassen vorhanden, die das Vertragsmodell realisieren (vgl. [Meyer97]).

5.1.1 Werkzeugkomponentenmanager

Der Werkzeugkomponentenmanager ist ein Teil des JWAM-Rahmenwerks. Er übernimmt die Verwaltung von Exemplaren von Werkzeugkomponenten. Der Werkzeugkomponentenmanager ist nach dem Singleton-Muster (vgl. [Gamma96]) realisiert. Nachfolgend wird die Realisierung des Werkzeugkomponentenmanagers beschrieben.

Am Werkzeugkomponentenmanager werden Exemplare von Werkzeugkomponenten angemeldet. Nach der Registrierung muss die Nachbedingung `isRegistered(tool)` gelten. Es werden zur Anmeldung keine Class-Objekte der Werkzeugkomponenten verwendet, da sie nicht die werkzeugspezifischen Informationen liefern können. Beispielsweise ist für die Erzeugung einer Sub-Werkzeugkomponente relevant, welche Funktionalität die angemeldeten Werkzeugkomponenten jeweils erfüllen. Der Typ der angebotenen Funktionalität kann über die *Tool*-Schnittstelle einer Werkzeugkomponente angefordert werden.

```
/**
   * @require tool != null
   * @require !has(tool)
   * @ensure has(tool)
   */
public void register (Tool tool);
```

Um eine Werkzeugkomponente abzumelden, muss diese bei dem Werkzeugkomponentenmanager registriert sein. Nach der Abmeldung darf die Werkzeugkomponente nicht mehr registriert sein.

```
/**
   * @require tool != null
   * @require has(tool)
   * @ensure !has(tool)
   */
public void unregister (Tool tool);
```

Mit folgender Methode können alle angemeldeten Werkzeugkomponenten abgemeldet werden.

```
public void unregisterAll ();
```

Außerdem kann mit der Operation

```
/**
   * @require tool != null
   */
public boolean isRegistered (Tool tool)
```

getestet werden, ob eine bestimmte Werkzeugkomponente bereits angemeldet ist.

Angemeldete Werkzeugkomponenten können von dem Manager erzeugt und auch für die Wiederverwendung in anderen Werkzeugkomponenten eingesetzt werden. Der Werkzeugkomponentenmanager sucht aus dem Pool der zur Verfügung stehenden Werkzeugkomponenten eine aus, die den angeforderten Typ erfüllt, und erzeugt ein neues Exemplar der Werkzeugkomponente und liefert dieses zurück. Die Erzeugung des Exemplars erfolgt über den Aufruf der Methode `newInstance()` des Class-Objektes der angemeldeten Werkzeugkomponente.

Zur Überprüfung, ob entsprechende Werkzeugkomponenten erzeugt werden können, stehen zwei Operationen zur Verfügung.

```
/**
   * @require toolType != null
   */
public boolean canCreateTool(Class toolType);


/**
   * @require functionalityType != null
   */
public boolean canCreateToolForFunctionality(Class
functionalityType);
```

Die erste Operation prüft, ob eine Werkzeugkomponente von dem übergebenen Typ erzeugt werden kann. Die zweite Operation untersucht, ob die Erzeugung einer Werkzeugkomponente, deren fachliche Funktionalität dem übergebenen Funktionalitäts-Typen entspricht, möglich ist. Dazu bietet der

Werkzeugkomponentenmanager zwei Operationen an, mit denen Werkzeugkomponenten erzeugt werden können.

```
/**
   * @require toolType != null
   * @require canCreateTool(toolType)
   * @ensure result != null
   */
public Tool newTool (Class toolType);

/**
   * @require toolClass != null
   * @require canCreateTool(toolClass)
   * @ensure result != null
   */
public Tool newToolForFunctionality (Class functionalityType);
```

Die `newTool (Class toolType)` Operation erzeugt eine Werkzeugkomponente von dem übergebenen Typ. Mit dem Aufruf der zweiten Operation kann eine Werkzeugkomponente angefordert werden, die den entsprechenden Typ der benötigten Funktionalität erfüllt. In beiden Operationen wird eine Werkzeugkomponente neu erzeugt. Das Exemplar der Werkzeugkomponente wird dann von dem Werkzeugkomponentenmanager verwaltet, bis die Werkzeugkomponente nicht mehr benötigt und geschlossen wird.

Die folgende Operation untersucht, ob das Exemplar einer Werkzeugkomponente, unter den laufenden Werkzeugkomponenten existiert. Im Unterschied zu `isRegistered`, prüft diese Methode nicht ob die Werkzeugkomponente angemeldet ist, sondern ob sie gerade läuft.

```
/**
   * @require tool != null
   */
public boolean existsTool (Tool tool);
```

Die Operation `existsTool (ToolFunctionality functionality)` prüft, ob ein Exemplar einer Werkzeugkomponente mit der übergebenen Funktionalität erzeugt wurde.

```
/**
   * @require functionality != null
   */
public boolean existsTool (ToolFunctionality functionality);
```

Verläuft eine der beiden Prüfungen positiv, so kann mit der entsprechenden Operation die bereits existierende Werkzeugkomponente angefordert werden.

Ferner liefert der Werkzeugkomponentenmanager alle sich in Benutzung befindenden Werkzeugkomponenten:

```
public Tool[] runningTools ();
```

oder auch alle gerade benutzten Werkzeugkomponenten, die einem bestimmten Typ entsprechen:

```
public Tool[] runningTools (Class toolType);
```

In der Ergebnismenge der beiden `runningTools` Operationen sind alle laufenden Kontext- und Sub-Werkzeugkomponenten enthalten.

Eine Werkzeugkomponente, die für die Erledigung ihrer Aufgaben Sub-Werkzeugkomponenten benötigt, wendet sich an den Werkzeugkomponentenmanager. In diesem Fall stehen zwei Operationen zur Auswahl. Die `newTool (Class toolType)` Operation kann verwendet werden, wenn bekannt ist, welche Werkzeugkomponente verwendet werden soll. Allerdings kann hier nicht garantiert werden, dass sich der Typ des Dienstes, also die Funktionalität dieser Werkzeugkomponente, nicht geändert hat. Daher sollte diese Operation nur für die Erzeugung von Kontext-Werkzeugkomponenten eingesetzt werden. Für die Erzeugung einer Sub-Werkzeugkomponente ist deren fachliche Funktionalität entscheidend. Der Typ der geforderten Funktionalität wird bei dem Aufruf der Operation `newToolForFunctionality (Class functionalityType)` übergeben. Der Werkzeugkomponentenmanager findet die passende Werkzeugkomponente, erzeugt diese und liefert sie dann zurück.

Beim Schließen einer Werkzeugkomponente wird der Werkzeugkomponentenmanager informiert, so dass er das Exemplar der geschlossenen Werkzeugkomponente aus seiner internen Liste der laufenden Werkzeuge entfernen kann.

5.1.2 Gegenstände in JWAM

Die Gegenstände[2] des WAM-Ansatzes werden in JWAM in einer gemeinsamen Schnittstelle *Thing* (Zeug) und ihrer Standard-Implementation *ThingImpl* grundlegend beschrieben. Alle Werkzeuge, Automaten und Materialien sollten die *Thing*-Schnittstelle implementieren, bzw. von *ThingImpl* erben. Nachfolgend ist die *Thing*-Schnittstelle abgebildet:

```
public interface Thing extends Cloneable, Serializable,
Atomizable, Identifiable
{
   public void rename (String newName);
   public String name ();
   public boolean equals (Object obj);
   public String typeDescription ();
   public dvThingDescription thingDescription ();
   …
}
```

Ein Zeug hat immer eine eindeutige Identifikation und einen Namen. Die Beschreibung des Typs kann über `typeDescription` angefordert werden. Ein Zeug liefert darüber hinaus seine detaillierte Beschreibung, in der weitere Informationen über den Gegenstand enthalten sind. Diese Informationen sind in dem Fachwert *dvThingDescription* definiert. Neben der Repräsentation, wie Icons oder Namen, sind hier die Informationen zu Attributen, deren Anzahl, Namen und Werte, angesiedelt. Eine Werkzeugkomponente wird über den Fachwert *dvToolDescription* beschrieben. Diese Klasse erbt von *dvThingDescription* und erweitert sie um Eigenschaften, die ein Werkzeug kennzeichnen.

Eine Werkzeugkomponente muss die Art von Materialien nennen, mit denen sie arbeiten kann. Ferner liefert sie Informationen über das Package, zu dem sie gehört,

[2] Ein Gegenstand ist etwas, mit dem Menschen umgehen. Werkzeuge, Materialien und Automaten sind Gegenstände (aus [Roock & Wolf 98]).

sowie die Bezeichnung für die Aufgabe, die ein Benutzer mit der Werkzeugkomponente erledigen kann. Die Beschreibung einer Werkzeugkomponente liefert auch Information über den Typ der Funktionalität, des fachlichen Dienstes, den sie anderen Werkzeugkomponenten zur Verfügung stellt.

5.1.3 Zustandsmodell der Werkzeugkomponenten

Werkzeuge und somit auch Werkzeugkomponenten können auf einer Arbeitsfläche (Desktop) vergegenständlicht werden. Lippert (vgl. [Lippert99]) schreibt dazu:

> *„Der Desktop (als Arbeitsfläche) kann dazu genutzt werden, auf ihm die WAM-Metaphern zu vergegenständlichen. Dazu können auf dem Desktop Werkzeuge, Materialien, Automaten und Behälter beispielsweise als benannte Icons präsentiert werden.“*

Für ein besseres Verständnis der Werkzeugkomponenten ist es sinnvoll, ein Zustandsdiagramm für eine Werkzeugkomponente zu erstellen. Solch ein Zustandsdiagramm erweist sich außerdem als große Implementierungshilfe, indem es den Entwickler bei der Modellierung der fachlichen Zustände behilflich ist und die Formulierung von Vor- und Nachbedingungen erleichtert. In Abbildung 18 haben wir das Zustandsdiagramm zu einer Werkzeugkomponente dargestellt.

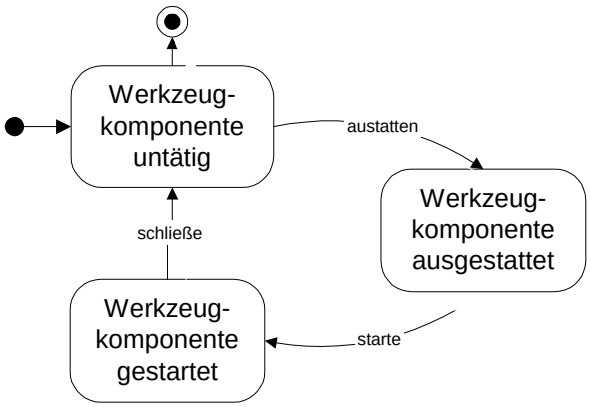

Abbildung 18: Zustandsdiagramm einer Werkzeugkomponente

Der Startzustand *Werkzeug untätig* bedeutet, dass eine Werkzeugkomponente auf dem Desktop liegt und auf Aktionen des Benutzers wartet. In diesem Zustand visualisiert der Desktop ein Exemplar der Werkzeugkomponente als Icon. Die Werkzeugkomponente ist noch nicht in der Lage, ihren fachlichen Dienst zu erfüllen. Technisch existiert nur das Hüllobjekt, ein Exemplar der Klasse `ToolMonoImpl` für monolithische und ein Exemplar der Klasse `ToolFpIpImpl` für Werkzeuge mit interner Trennung in eine Funktionskomponente und Interaktionskomponente. Durch eine Aktion des Benutzers, wie etwa das Ablegen eines Materials auf das Werkzeug-Icon beispielsweise über Drag&Drop, wird die Werkzeugkomponente aktiviert. Sie muss noch die zum Start notwendigen Teilstrukturen erzeugen. Wie wir Abbildung 19 zeigen, muss eine Werkzeugkomponente mit interner Trennung zwischen Funktion und Interaktion noch zusätzliche Zustände durchlaufen, bevor sie in den für das Starten notwendige Zustand *Werkzeugkomponente ausgestattet* gelangen kann. Als erstes wird die eigene Funktionalität und die Benutzungsoberfläche erzeugt, d.h. die

Werkzeugkomponente erzeugt Exemplare der internen Funktions- und Interaktionsklassen. Somit hat die Werkzeugkomponente ihre eigene innere Struktur aufgebaut. Die Werkzeugkomponente geht somit in den Zustand *Werkzeugkomponenten-Rumpf erzeugt* über.

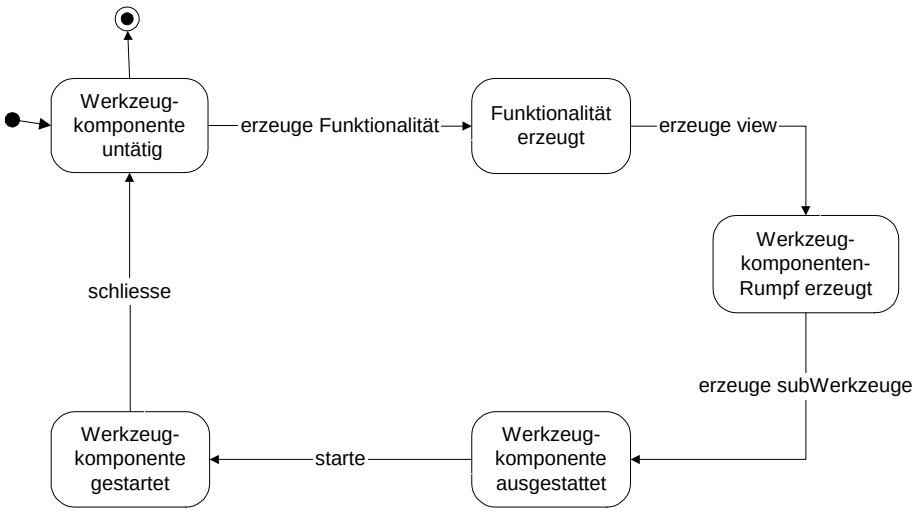

Abbildung 19: Zustandsdiagramm einer Werkzeugkomponente mit Trennung zwischen Funktionalität und Interaktion

Nachdem der Werkzeug-Rumpf vollständig ist, müssen noch eventuell benötigte Sub-Werkzeuge erzeugt werden, um die Werkzeugkomponente starten zu können. So gelangt die Werkzeugkomponente in den Zustand *Werkzeugkomponente ausgestattet*. Der Anwender kann das Werkzeug schließen, wodurch die Werkzeugkomponente wieder in den Zustand *Werkzeugkomponente untätig* gelangt.

5.1.4 Werkzeugkomponenten und innere Struktur

Eine Werkzeugkomponente enthüllt nach außen keine Details über ihren Aufbau. Die im Kapitel 3 beschriebene Trennung zwischen Funktionalität und Interaktion eines Werkzeugs hat sich bei der Entwicklung von Softwarewerkzeugen bewährt. Es hat sich gezeigt, dass für den Bau eines Testwerkzeugs oder eines Prototypen, die in Kapitel 3 beschriebene Konstruktion zu zeitaufwendig ist. Deshalb haben wir eine neue Möglichkeit zur Erstellung von Werkzeugkomponenten zum JWAM-Rahmenwerk hinzugefügt. Wir unterstützen zwei unterschiedliche Arten von Werkzeugkomponenten: monolithische Werkzeugkomponenten (*ToolMonoImpl*), für die schnelle prototypische Erstellung von Werkzeugkomponenten, und Werkzeugkomponenten mit Trennung von Funktionalität und Interaktion (*ToolFPIPImpl*).

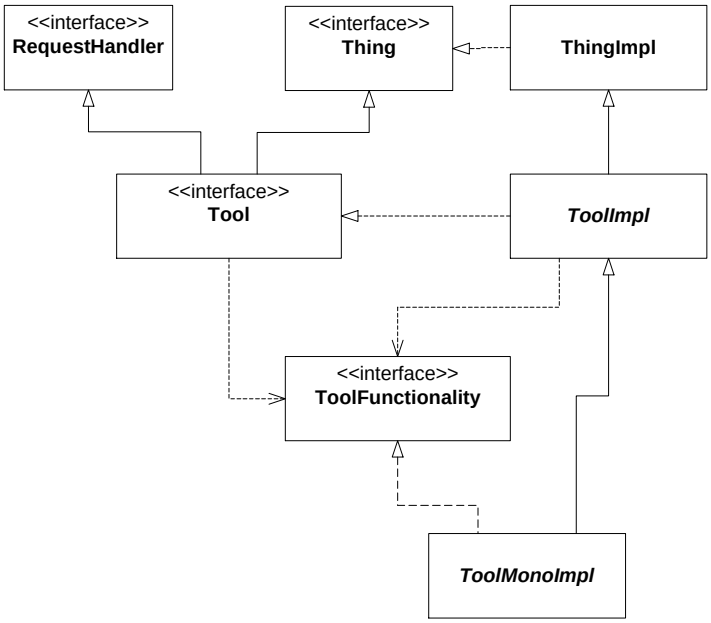

Abbildung 20: Klassendiagramm: Monolithische Werkzeugkomponenten

Die *ToolInteraction*-Klasse spielt nur eine Rolle für Werkzeugkomponenten mit der Trennung zwischen Funktionalität und Interaktion. In Abbildung 20 ist ein Klassendiagramm für monolithische Werkzeugkomponenten dargestellt. Eine monolithische Werkzeugkomponente beinhaltet keine Trennung zwischen Funktionalität und Interaktion. Da auch eine monolithische Werkzeugkomponente die *Tool*-Schnittstelle erfüllen muss, erbt die Klasse *ToolMonoImpl* einerseits von *ToolImpl* und implementiert andererseits die Schnittstelle *ToolFunctionality*. Damit ist gewährleistet, dass eine monolithische Werkzeugkomponente über ihre *Tool*-Schnittstelle die fachliche Funktionalität anderen Werkzeugkomponenten zur Verfügung stellen kann.

Die innere Struktur kann bei monolithischen Werkzeugkomponenten beliebig gewählt werden.

In Abbildung 21 haben wir die Klassen des Rahmenwerks für die Erstellung einer Werkzeugkomponente mit der Trennung zwischen Funktionalität und Interaktion dargestellt.

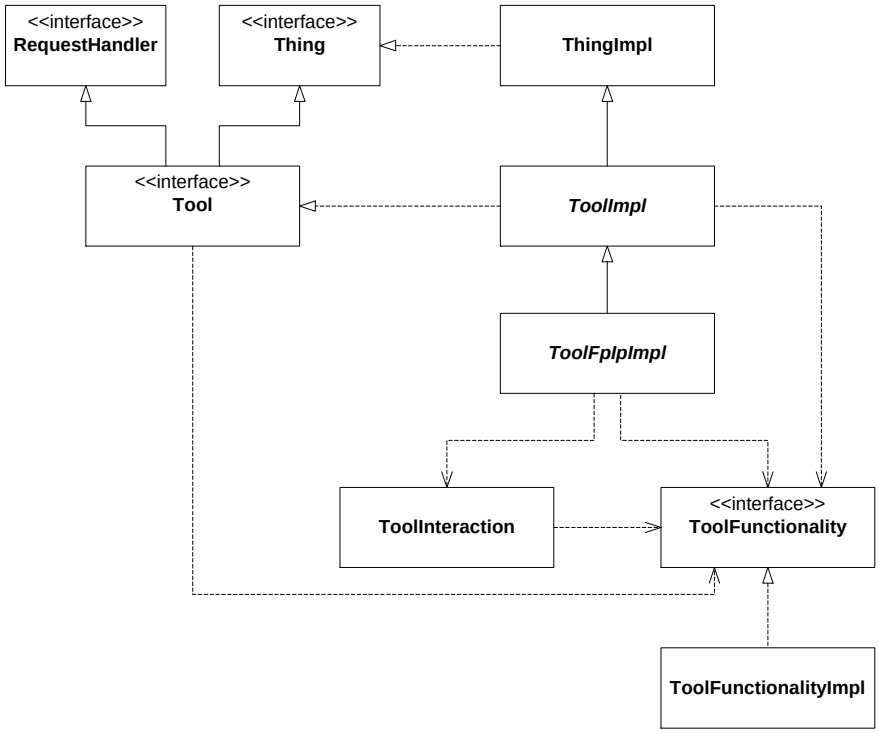

Abbildung 21: Klassendiagramm: Werkzeugkomponenten mit Trennung zwischen Funktionalität und Interaktion

Die Entwicklung von Werkzeugkomponenten mit Trennung zwischen Funktionalität und Interaktion wird durch die abstrakte Klasse *ToolFPIPImpl* unterstützt. Wie in Abbildung 21 dargestellt, erbt diese Klasse von *ToolImpl* und benutzt für die Realisierung der Interaktion die Klasse *ToolInteraction* und für die fachliche Funktionalität die Schnittstelle *ToolFunctionality*.

Nach außen verhalten sich beide Arten von Werkzeugkomponenten gleich. Die einheitliche Behandlung von allen Werkzeugkomponenten erleichtert und ermöglicht bessere Wiederverwendbarkeit. Da alle Werkzeugkomponenten die Schnittstellen *Tool*, *Thing* und *RequestHandler* besitzen, sowie ihren *GUIContext* und vor allem ihre fachliche Funktionalität in Form der *ToolFunctionality*–Schnittstelle zur Verfügung stellen, gibt es bei der Wiederverwendung keine Kompatibilitätsprobleme. Darüber hinaus erleichtert der Werkzeugkomponentenmanager die Erzeugung und Verwendung von Werkzeugkomponenten.

In Abbildung 22 zeigen wir mit Hilfe eines Interaktionsdiagramms die Erzeugung einer Werkzeugkomponente. Die einzelnen Werkzeugkomponenten müssen zuerst bei dem Werkzeugkomponentenmanager registriert werden. Ein Exemplar einer Werkzeugkomponente wird über den Werkzeugkomponentenmanager angefordert.

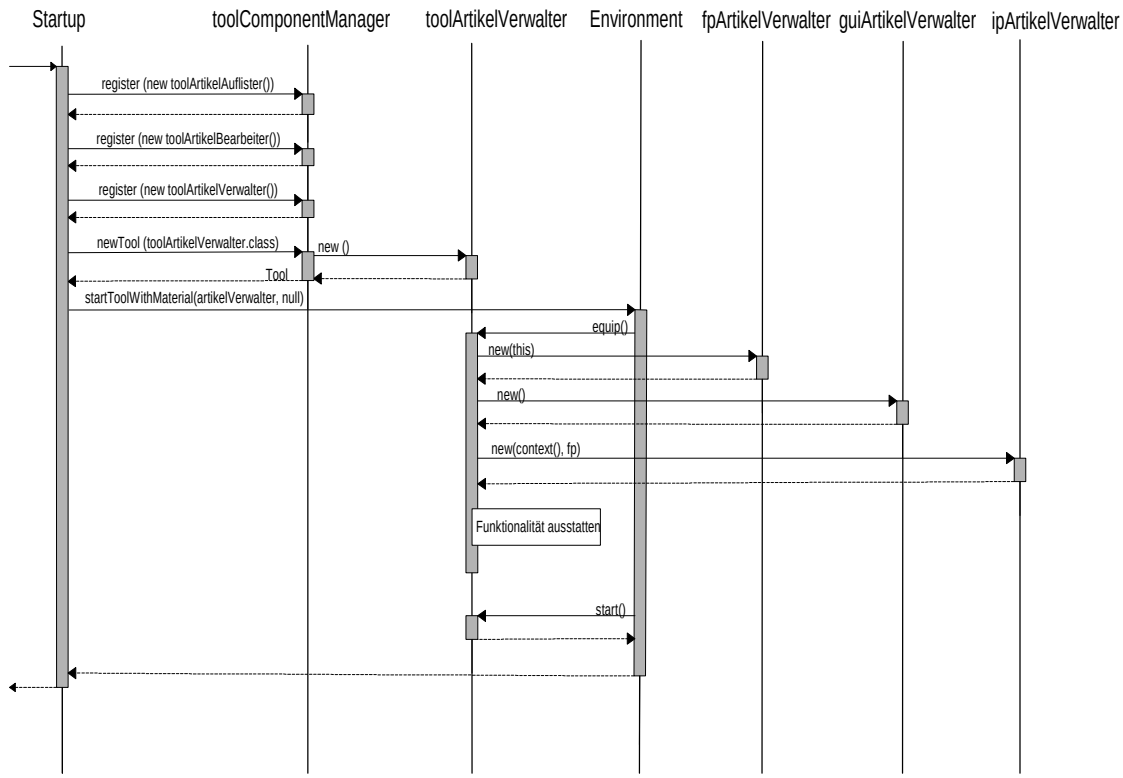

Abbildung 22: Erzeugung einer Werkzeugkomponente

Die Schnittstelle *Tool* definiert Operationen, die jede Werkzeugkomponente implementieren muss. Sie erbt von den beiden Schnittstellen *Thing* und *RequestHandler*.

```
public interface Tool extends Thing, RequestHandler
{
     public boolean canStartWithoutMaterial ();
     public void start ();
     public void start (Thing material);
     public void stop ();
     public boolean isRunning ();
     public void setMaterial (Thing material);
     public ToolFunctionality functionality ();
     public boolean isEquipped ();
     public void equip ();
     public void equip (Tool parentTool);
     public void unequip ();
     public GUIContext context ();
     public boolean hasContext ();
     public Tool[] subTools ();
     public dvToolDescription toolDescription ();
}
```

Die Werkzeugkomponentenschnittstelle *Tool* bietet Operationen zum Starten und Stoppen des Werkzeugs. Nachdem eine Werkzeugkomponente erzeugt wurde, wird sie ausgestattet, d.h. mit der Operation `equip ()` kann eine Werkzeugkomponente als Kontext-Werkzeugkomponente eingerichtet werden, wobei die erforderlichen Sub-Werkzeugkomponenten erzeugt und eingebunden werden (siehe Abschnitt 5.1.3). Die Operation `equip (Tool parentTool)` richtet die Werkzeugkomponente als eine Sub-Werkzeugkomponente ein. Nach dem Aufruf der entsprechenden `equip` Operation ist die Werkzeugkomponente ausgestattet und kann nun gestartet werden. Das Starten geschieht über eine der Operationen `start()` bzw. `start (Thing material)`. In dem ersten Fall wird ein Werkzeug ohne ein Material gestartet, im zweiten Fall mit einem Material. Die Prüfung, ob eine Werkzeugkomponente gestartet wurde kann mit der Operation `isRunning ()` durchgeführt werden. Der Aufruf der `stop ()` Operation stoppt das Werkzeug. Wenn die Werkzeugkomponente gestoppt wurde, kann die Ausstattung zurückgenommen werden (`unequip ()`).

Entweder kann ein Material beim Starten des Werkzeugs mit der Methode `start (Thing material)` oder über die Operation `setMaterial (Thing material)` gesetzt werden.

Die Schnittstelle *Tool* bietet über die Operation `functionality ()` den Zugriff auf die fachliche Funktionalität. Eine Werkzeugkomponente kann von einer anderen Werkzeugkomponente über den Aufruf dieser Operation die Funktionalität anfordern und ihre eigene fachliche Funktionalität erweitern. Die Funktionalität einer Werkzeugkomponente hat immer einen bestimmten Typ. Die Beschreibung einer Werkzeugkomponente *dvToolDescription* liefert Informationen über den Typ des Dienstes, der angeboten wird.

Eine zusammengesetzte Werkzeugkomponente besteht aus mehreren Sub-Werkzeugkomponenten, die zur Erledigung der Aufgabe beitragen. Die Operation `subTools ()` liefert alle vorhandenen Sub-Werkzeugkomponenten, die in einer Kontext-Werkzeugkomponente eingebunden sind.

Die Benutzungsoberfläche einer Werkzeugkomponente kann über die Operation `context ()` angefordert werden. Über den *GUIContext* kann eine Werkzeugkomponente in die Benutzungsoberfläche ihrer Kontext-Werkzeugkomponente eingebunden werden. Eine andere Möglichkeit besteht darin, dass die Sub-Werkzeugkomponente über diese Operation den *GUIContext* ihrer Kontext-Werkzeugkomponente anfordert, um diesen mitzubenutzen.

Das Rahmenwerk bietet mit der Klasse *ToolImpl* eine Standard-Implementation der Werkzeugkomponentenschnittstelle *Tool* an. Es handelt sich dabei um eine abstrakte Klasse, die die Operationen der Tool-Schnittstelle bis auf die `functionality()` Operation implementiert und andererseits von der Klasse *ThingImpl* erbt. Durch diese Vererbungshierarchie übernimmt *ToolImpl* die Standard-Implementation der *Thing*-Schnittstelle und muss die definierten Operationen nicht selbst implementieren. Anders verhält es sich mit der *RequestHandler*-Schnittstelle. Die Operationen dieser Schnittstelle müssen noch implementiert werden. Dabei handelt es sich um eine Operation:

```
public void handle (Request req);
```

In der Standard-Implementation dieser Operation wird nach der Überprüfung der Vorbedingungen eine Einschubmethode `doHandle (Request req)` aufgerufen. Diese Einschubmethode bietet die Möglichkeit, in einer konkreten

Werkzeugkomponente die Behandlung von Anfragen zu definieren. Die Standard-Implementation bietet die Behandlung einer Anfrage für das Erzeugen von Sub-Funktionalität und damit das Erzeugen von Sub-Werkzeugkomponenten an. Hier wird zunächst geprüft, ob die Anfrage bereits behandelt wurde. Wenn dies nicht der Fall ist und die Anfrage vom Typ *reqCreateFunctionality* ist, dann wendet sich die Werkzeugkomponente an den Werkzeugkomponentenmanager mit der Frage, ob er eine Werkzeugkomponente, deren Funktionalität benötigt wird, erzeugen kann. Die Information über den Typ der erforderlichen Funktionalität liefert das Anfrageobjekt. Über den Aufruf der Operation `newToolForFunctionality (Class functionalityType)` fordert die Werkzeugkomponente eine neue Sub-Werkzeugkomponente vom Werkzeugkomponentenmanager an. Die Sub-Werkzeugkomponente wird dann ausgestattet und ihre Funktionalität wird bei dem Anfrageobjekt gesetzt, so dass nach dem Behandeln der Anfrage, die Funktionalität direkt von dem Anfrageobjekt erhalten werden kann. Folgender Quellcode enthält die beschriebene Behandlung einer *reqCreateFunctionality*-Anfrage:

```
if (!req.isHandled() && req instanceof reqCreateFunctionality)
{
   Class functionalityType =
      ((reqCreateFunctionality)req).functionalityType();
   if (ToolComponentManager.instance().
      canCreateToolForFunctionality(functionalityType))
   {
      Tool tool = ToolComponentManager.instance().
         newToolForFunctionality(functionalityType);
      tool.equip(this);
      ToolFunctionality toolFunctionality =
         tool.functionality();
      ((reqCreateFunctionality)req).
         setFunctionality(toolFunctionality);
   }
}
```

Die fachliche Funktionalität einer Werkzeugkomponente hat einen bestimmten Typ. Daher muss die Operation `functionality()` von jeder konkreten Werkzeugkomponente selbst implementiert werden. Über diese Methode wird die fachliche Funktionalität der Werkzeugkomponente geliefert.

Die Ausstattung einer Werkzeugkomponente ist in den beiden `equip` Operationen festgelegt. Jede dieser Operationen enthält den Aufruf einer abstrakten Einschubmethode, die von einer konkreten Werkzeugkomponente implementiert werden muss. Eine Werkzeugkomponente ist ausgestattet, wenn sie einen *RequestHandler* hat und ihre Funktionalität ausgestattet ist. Der *RequestHandler* kann die Umgebung (*Environment*) oder eine andere Werkzeugkomponente sein. Die fachliche Funktionalität einer Werkzeugkomponente kann Erweiterungen durch Funktionalität einer oder mehreren Sub-Werkzeugkomponenten erfordern. Diese zusätzliche Funktionalität kann, je nach Bedarf, entweder im Rahmen der Ausstattung der Kontext-Werkzeugkomponente oder auch dynamisch[3] erzeugt und eingebunden werden. Die Operation `equip ()` richtet eine Werkzeugkomponente

[3] z.B. beim Betätigen einer Schaltfläche durch den Benutzer

als Kontext-Werkzeugkomponente ein. In der Einschubmethode `doEquip ()` kann die Erzeugung der konkreten Funktionalität, der Interaktion und der Benutzungsoberfläche des Werkzeugs definiert werden. Die `equip (Tool parentTool)` Operation richtet dagegen eine Werkzeugkomponente als Sub-Werkzeugkomponente ein. Wiederum wird eine Einschubmethode `doEquip (Tool parentTool)` implementiert, um die Erzeugung der eigenen Funktionalität, Interaktion und der Benutzungsoberfläche festzulegen. Eine Sub-Werkzeugkomponente kann ihre eigene Benutzungsoberfläche nutzen. Sie kann aber auch die Benutzungsoberfläche von ihrer Kontext-Werkzeugkomponente anfordern (über `parentTool.context()`) und ihre Interaktion an diese binden.

Die Ausstattung der Werkzeugkomponente kann rückgängig gemacht werden. Zu diesem Zweck dient die Operation `unequip ()`, die zwar eine Standard-Implementation in der Klasse *ToolImpl* enthält, jedoch ebenfalls mit einer Einschubmethode `doUnequip()` versehen ist. Die Einschubmethode muss von einer konkreten Werkzeugkomponente entsprechend implementiert werden.

Die Schnittstelle *ToolFunctionality* definiert Operationen der Funktionalität einer Werkzeugkomponente. Die *ToolFunctionality*-Schnittstelle sieht wie folgt aus:

```
public interface ToolFunctionality extends EventSubject
{
    public String name ();
    public void rename (String name);
    public void tryToClose ();
    public void equip ();
    public void unequip ();
    public boolean isEquipped ();
    public void setMaterial (Thing material);
    public void useMaterial (Thing material);
    public void unuseMaterial (Thing material);
    public boolean hasMaterial (Thing material);
    public void updateMaterial (Thing material);
    public Thing[] materials ();
    public ToolFunctionality[] subFunctionalities ();
}
```

Die Operation `name ()` liefert den Namen der Funktionalität. Diesen Namen kann man mit der Operation `rename (String name)` überschreiben. Mit dem Aufruf der `tryToClose ()` Operation wird versucht, die Funktionalität der Werkzeugkomponente zu schließen. Dabei wird ein *reqCloseTool*-Anfrageobjekt erzeugt und entlang der Zuständigkeitskette an den *RequestHandler* übergeben.

Die Funktionalität einer Werkzeugkomponente kann über die Operation `equip ()` ausgestattet werden. Dabei können beispielsweise Sub-Funktionalitäten hinzugefügt werden. Die Operation `isEquipped ()` prüft, ob die Funktionalität ausgestattet wurde. Die Funktionalität einer Werkzeugkomponente kann alle verwendeten Sub-Funktionalitäten liefern (`subFunctionalities ()`). Die *ToolFunctionality*-Schnittstelle definiert darüber hinaus Operationen zum Umgang mit Materialien.

Die Klasse *ToolFunctionalityImpl*, die im JWAM-Rahmenwerk angeboten wird, realisiert eine Standard-Implementation der Schnittstelle *ToolFunctionality*. Diese

Klasse entspricht dem Konzept der Funktionskomponente. Bei der Entwicklung einer Werkzeugkomponente sollte deren fachliche Funktionalität von dieser Klasse erben, um nur die notwendigen Operationen überschreiben zu müssen. Dieses Vorgehen erleichtert die Entwicklung neuer Funktionalitäten, da sich der Entwickler auf die Implementation der fachlichen Funktionalität konzentrieren kann.

Die Klasse *ToolInteraction* realisiert die Interaktion einer Werkzeugkomponente. *ToolInteraction* übernimmt die Aufgaben einer Interaktionskomponente. Durch Erben von dieser Klasse können konkrete Interaktionskomponenten erzeugt werden. *ToolInteraktion* arbeitet mit genau einem Exemplar einer Werkzeugkomponenten-Funktionalität. Die Implementation der Klasse *ToolInteraction* enthält Operationen zum Anzeigen der Benutzungsoberfläche auf dem Bildschirm, sowie zum Ausblenden und Schließen der Interaktion. Außerdem kann die Interaktion die entsprechende Funktionalität der Werkzeugkomponente, mit der sie verknüpft ist, zurückliefern. Sie ermöglicht die Prüfung, ob ein *GUIContext* vorhanden ist und liefert diesen auf Anfrage zurück.

5.1.5 Die Kommunikation der Werkzeugkomponenten

Die Kommunikation innerhalb einer Werkzeugkomponente mit Trennung zwischen Funktionalität und Interaktion erfolgt über Operationsaufrufe, Ereignisse und Anfragen. Die Kommunikation ist schematisch in Abbildung 23 für die Artikelauflister-Werkzeugkomponente dargestellt.

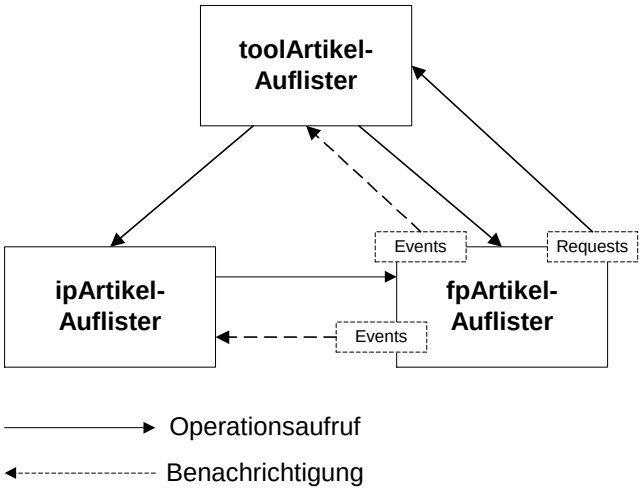

Abbildung 23: Interne Kommunikation einer Werkzeugkomponente

Die Funktionalitätsklasse *ToolFunctionalityImpl* bietet an ihrer Schnittstelle Ereignisse für fachliche Zustandsübergänge an. In Abbildung 24 zeigen wir die Zusammenhänge der Ereignis-Benachrichtigung und deren Einordnung in das Rahmenwerk auf. Die fachlichen Zustände einer Werkzeugkomponente sind in der Funktionalität festgehalten. Die Oberklasse aller Funktionalitäten *ToolFunctionalityImpl* erbt von der Klasse *EventSubjectImpl*, mit der eine Implementierung der Schnittstelle *EventSubject* vorliegt. Wenn eine Zustandsänderung auftritt, kann die Funktionskomponente ihre registrierten Beobachter benachrichtigen. Die Interaktionskomponente benutzt die

EventReaction-Schnittstelle. In der `update ()`-Operation legt die Interaktionskomponente das Verhalten für eine fachliche Zustandsänderung der Funktionskomponente fest.

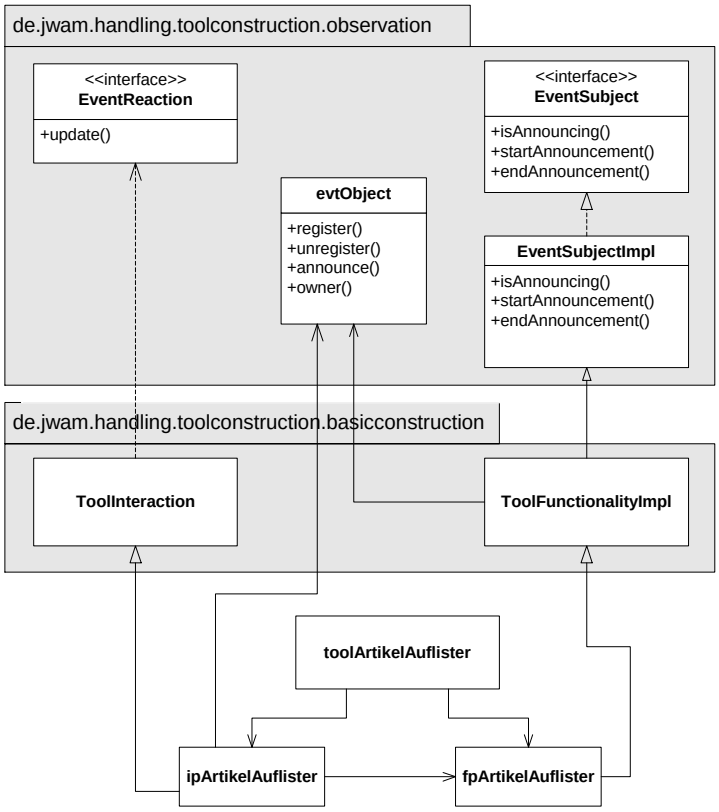

Abbildung 24 : Ereignis-Beobachter-Muster Implementierung in JWAM

Neben den Ereignissen können Funktionskomponenten Anfragen (*requests*) an ihre Werkzeugkomponente richten. Diese Anfragen sind dann nötig, wenn die Funktionskomponente einen Dienst nicht eigenständig erfüllen kann und ihren Kontext, in diesem Fall die Werkzeugkomponente, darüber informiert. Das Klassendiagramm in Abbildung 25 zeigt, dass die Umgebung und jedes Werkzeug die Schnittstelle *RequestHandler* erfüllen, und somit als Anfragen-Bearbeiter in die Zuständigkeitskette aufgenommen werden können. Der Konstruktor der Klasse *ToolFunctionalityImpl* erwartet als Parameter den *RequestHandler*:

```
public ToolFunctionalityImpl (RequestHandler handler).
```

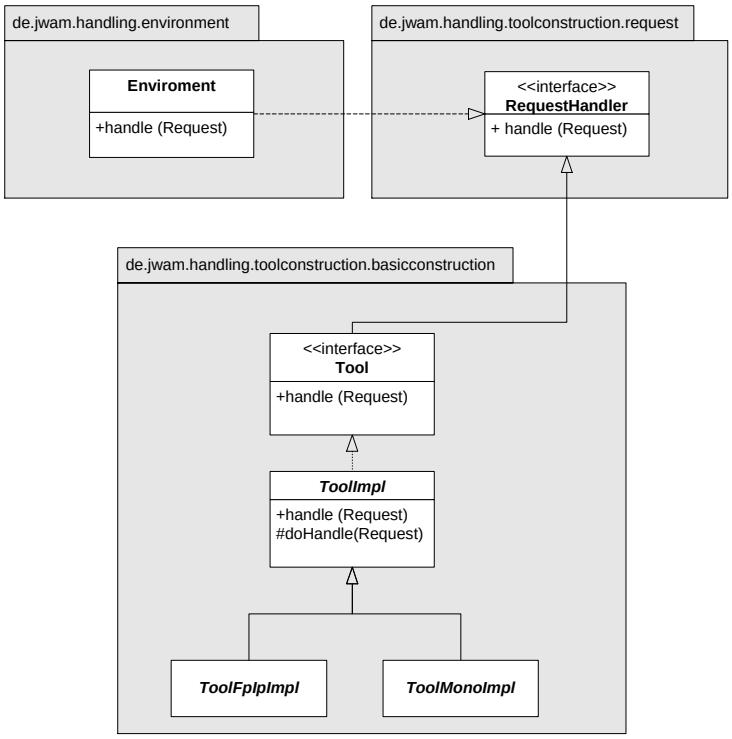

Abbildung 25:Implementierung der Zuständigkeitskette in JWAM

Somit kann sich die Werkzeugkomponente bei der Erzeugung der Funktionskomponente als deren *RequestHandler* registrieren und in der Zuständigkeitskette zur Verfügung stehen. Nachfolgender Codeabschnitt zeigt das Erzeugen der Funktionskomponente für die ArtikelAuflister-Werkzeugkomponente:

```
protected void doEquip ()
{
   ...
   fpArtikelAuflister fp = new fpArtikelAuflister( this );
   ...
}
```

Im Moment gibt es folgende Fälle, in denen eine Anfrage entlang der Zuständigkeitskette aktiviert werden muss:

- Das Werkzeug soll erzeugt werden.
- Das Werkzeug soll geschlossen werden.
- Das Material soll gespeichert werden.
- Eine Sub-Funktionalität soll erzeugt werden.

Wenn einer dieser Aktionen auftritt, erzeugt die Funktionalität ein Exemplar einer entsprechenden *Request*-Klasse aus Abbildung 26. Die Operation `handle (Request req)` des *RequestHandlers*, die als Parameter ein Objekt vom Typ *Request* erwartet wird mit dem erzeugten Anfrage-Objekt aufgerufen.

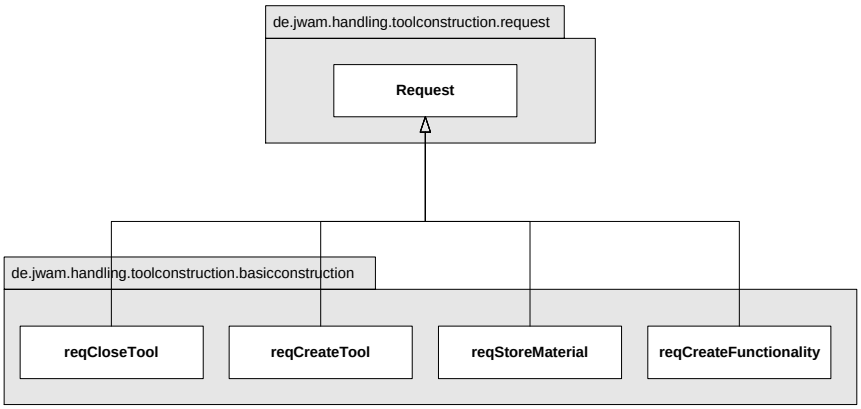

Abbildung 26: *Request*-Klassen für die Werkzeugkomponenten

Die Klasse *Environment* bildet den obersten *RequestHandler* in der Zuständigkeitskette. Durch die in Abbildung 26 dargestellte Lösung mit der Unterklassen-Bildung, sieht die `handle (Request req)` Operation der *Environment*-Klasse folgendermaßen aus:

```
public final void handle (Request req)
{
   Contract.require(req != null, this, "req not null");
   Contract.require(!req.isHandled(), this, "request not handled "
      + req);
   if (req instanceof reqCreateTool)
   {
      handleCreateToolRequest((reqCreateTool) req);
   }
   else if (req instanceof reqCloseTool)
   {
      handleCloseRequest((reqCloseTool) req);
   }
   else if (req instanceof reqStoreMaterial)
   {
      handleStoreMaterialRequest((reqStoreMaterial) req);
   }
}
```

Die Klasse *Environment* muss in einem `if`-Konstrukt abfragen, um welchen *Request*-Typ es sich handelt. Da sich in diesem Fall die Anzahl der *Request*-Typen noch in Grenzen hält, kann diese Lösung gewählt werden. Sollte die Anzahl der Anfragen allerdings steigen, müsste man über eine übersichtlichere Lösung nachdenken.

Zur Kommunikation zwischen Kontext- und Sub-Werkzeugkomponenten bedienen sich die Werkzeugkomponenten, analog zu der Kommunikation innerhalb einer

Werkzeugkomponente, der in JWAM realisierten Muster für den Ereignis-Beobachter und der Zuständigkeitskette.

Eine Kontext-Werkzeugkomponente wird über fachliche Zustandsänderungen ihrer Sub-Werkzeugkomponente informiert. Die Funktionalität der Kontext-Werkzeugkomponente meldet sich als Beobachter bei der Funktionalität der Sub-Werkzeugkomponente, analog dem oben beschriebenen Verfahren, an. Danach kann die Funktionalität der Sub-Werkzeugkomponente bei einem fachlichen Zustandsübergang ihren Beobachter, die Funktionalität der Kontext-Werkzeugkomponente, informieren. Diese kann dann ihrerseits über ein Ereignis ihre Interaktion benachrichtigen, so dass eine eventuelle Anpassung der Darstellung erfolgen kann.

In Abbildung 27 stellen wir die Erzeugung von Sub-Werkzeugkomponenten und den Aufbau der Zuständigkeitskette dar. Eine Kontext-Werkzeugkomponente registriert sich als *RequestHandler* bei der Sub-Werkzeugkomponente. Dadurch kann die Sub-Werkzeugkomponente die von ihrer Funktionalität stammenden Anfragen entlang der Zuständigkeitskette an das Kontext-Werkzeug weiterleiten, falls es sich um eine Anfrage handelt, die sie nicht bearbeiten kann.

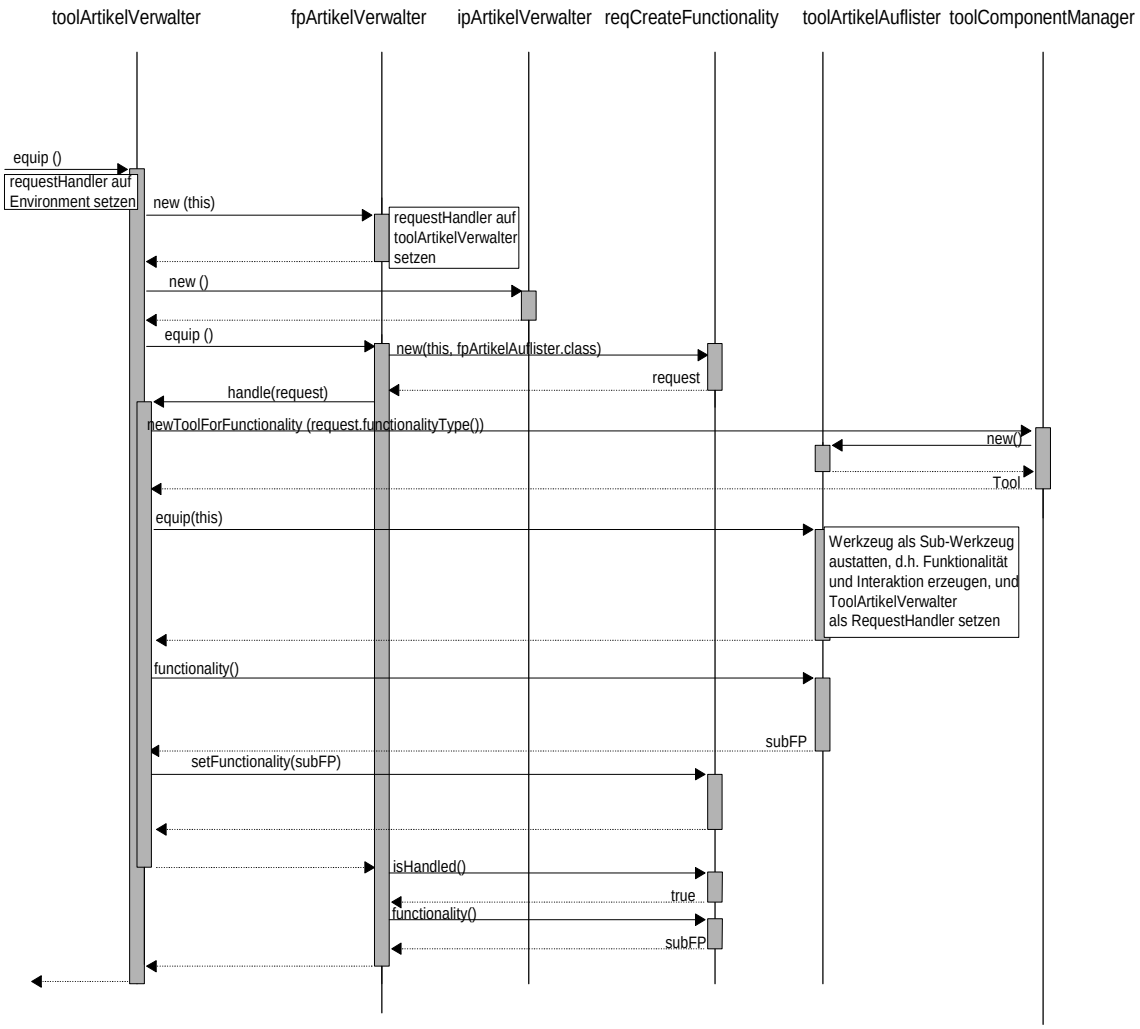

Abbildung 27: Erzeugung von Sub-Werkzeugkomponenten und Aufbau der Zuständigkeitskette

5.2 Abgrenzung zu Java-Beans

JavaBeans sind eine Erweiterung der Sprache Java um ein Komponentenmodell. Der Komponentenbegriff wird von JavaSoft (vgl. [Sun97]) wie folgt definiert:

> *„JavaBeans components, or Beans, are reusable software components that can be manipulated visually in a builder tool."*

Die Definition ist bezüglich der Größe und des Einsatzspektrums einer JavaBeans-Komponente recht weit gefasst. Das Spektrum reicht von einfachen Schaltflächen für Oberflächen bis hin zu komplexen Komponenten wie Textverarbeitungs- oder Tabellenkalkulationskomponenten. Obwohl in der Regel mit JavaBeans „visuelle Komponenten" realisiert werden, sind sie nicht notwendigerweise sichtbar, d.h. sie können auch keine GUI-Represäntation haben. JavaBeans bestehen aus mehreren zusammenhängenden Klassen, eventuell aus einer Informationsklasse, zur Auswertung durch spezielle IDEs und einer Dokumentation in Form von Html-Dateien. Als Programmierkonstrukt sind JavaBeans „normale" Java-Klassen. D.h. es gibt keine allgemeine Hauptklasse für Komponenten, von der alle JavaBeans erben müssten. Java-Klassen werden zu JavaBeans, indem bestimmte Verhaltens-

und Namenskonventionen einhalten. Die folgenden Schlagwörter charakterisieren das Java-Komponentenmodell:

- Ereignisse,
- Eigenschaften,
- Methoden,
- Introspektion,
- Persistenz.

Für eine detaillierte Beschreibung des JavaBeans-Komponentenmodells verweisen wir auf [Sun97].

Das Komponentenmodell von JavaBeans ist implementierungsbezogen und als eine Erweiterung der Sprachfähigkeiten von Java anzusehen. Das von uns vorgestellte Modell von Werkzeugkomponenten legt den Fokus auf die Architektur- und Modellierungsebene. Die in diesem Kapitel vorgestellte Realisierung des Konzeptes beruht nicht auf JavaBeans. Im ersten Schritt der Realisierung wollten wir eine möglichst einfache Integration des Konzepts in das vorhandene JWAM-Rahmenwerk vornehmen. Eine Realisierung in JavaBeans hätte nur dann Sinn, wenn die Werkzeugkomponenten in einem Entwicklungstool dargestellt und bearbeitet würden, was bei unserem jetzigen Realisierungsstand noch nicht gegeben ist. Wir sehen JavaBeans als eine mögliche Implementierungsart für Werkzeugkomponenten an, d.h. eine Werkzeugkomponente, wie sie zur Zeit realisiert wird, könnte zu einem JavaBean modifiziert werden. Allerdings müsste noch detailliert untersucht werden, welche Fragestellungen und Probleme sich aus solch einem Einsatz ergeben würden.

5.3 Zusammenfassung

In diesem Kapitel haben wir die Realisierung des komponentenorientierten Werkzeug-Konzeptes für das JWAM-Rahmenwerk erläutert.

Die Trennung zwischen Funktionskomponente und Interaktionskomponente wird nicht mehr erzwungen, da neben einer Werkzeugkomponente, deren Struktur diese Trennung aufweist, monolithische Werkzeugkomponenten unterstützt werden. Monolithische Werkzeugkomponenten eignen sich besonders für den Bau von Werkzeugkomponenten-Prototypen. Der Werkzeugkomponentenmanager ermöglicht Flexibilisierung und Vereinheitlichung der Erzeugung von Werkzeugen.

Der Werkzeugkomponentenmanager kann unserer Meinung nach die Grundlage für die Werkzeugentwicklung mit Hilfe eines visuellen Entwicklungstools für nach dem WAM-Ansatz entworfene Anwendungen darstellen.

6 Abschluss und Ausblick

6.1 Zusammenfassung der Ergebnisse

Objektorientierung und Rahmenwerke bilden die Grundlage für komponentenorientierte Softwareentwicklung. Der Einsatz der komponentenorientierten Softwareentwicklung erhöht die Wiederverwendbarkeit und reduziert die Komplexität von großen Softwaresystemen. Die Verwendung vorgefertigter und ausgetesteter Komponenten in der Anwendungsentwicklung erhöht die Qualität der Software. Das Kapseln von zusammenhängenden Klassenstrukturen zu Komponenten in Rahmenwerken vereinfacht das Erlernen der Anwendungsentwicklung mit dem Rahmenwerk.

Wir haben das Konzept der Werkzeugkonstruktion, das in der Version 1.4 des JWAM-Rahmenwerks umgesetzt ist, beschrieben. Unsere Motivation für Veränderungen an dem Konzept war in erster Linie die Komplexität des Rahmenwerks, welche innerhalb kürzester Zeit stark zugenommen hat. Die Folge aus dieser Entwicklung ist, dass es für den Anwendungsentwickler und auch für den Rahmenwerksentwickler schwer fällt, das ganze Rahmenwerk zu überblicken. Der Bereich der Werkzeugkonstruktion bot sich uns an, um Beispielhaft zu untersuchen, wie die Komponentenorientierung auf Basis von Rahmenwerken helfen kann die Komplexität zu reduzieren.

In Kapitel 2 haben wir unser Verständnis einer komponentenorientierten Softwareentwicklung und ihre Beziehung zur klassischen objektorientierten Anwendungsentwicklung mit Rahmenwerken erarbeitet. Auf dieser Grundlage sind wir zu der Schlussfolgerung gelangt, dass die Objektorientierung und Rahmenwerke die Grundlage für komponentenorientierte Softwareentwicklung bilden. Komponentenorientierung und die Anwendungsentwicklung mit Rahmenwerken auf Basis der Objektorientierung ergänzen sich gegenseitig. Auf der einen Seite bietet das Rahmenwerk den Komponenten den benötigten architektonischen Rahmen. Auf der anderen Seite wird die Komplexität beim Einsatz und der Pflege des Rahmenwerks durch den Einsatz von Komponenten reduziert.

Durch Wiederverwendung einer bewährten, in einem Anwendungskontext ausgetesteten und eingesetzten Komponente, erhöht sich deren Qualität, und somit die Qualität der Anwendung insgesamt. Durch die Abstraktion von mehreren miteinander kooperierenden Klassen in Komponenten wird die Komplexität reduziert, so dass das Erlernen und die Überschaubarkeit des Rahmenwerks für Anwendungsentwickler einfacher wird.

In dieser Arbeit haben wir auf der Basis der Diplomarbeit von Fröse (vgl. [Fröse99]) ein Konzept für Werkzeugkomponenten nach dem WAM-Ansatz erarbeitet. Wir behandeln Subwerkzeuge und Kontextwerkzeuge einheitlich als Werkzeugkomponenten. Eine Werkzeugkomponente verbirgt die Details über ihren Aufbau, die interne Trennung zwischen Funktionalität und Interaktion ist nicht mehr sichtbar und auch nicht mehr zwingend erforderlich. Der Bau von Werkzeugkomponenten-Prototypen wird durch monolithische Werkzeugkomponenten unterstützt. Durch die Einführung des

Werkzeugkomponentenmanagers erhöht sich die Flexibilität und Effizient bei der Werkzeugentwicklung.

Das erarbeitete Konzept wurde in die Version 1.5 des JWAM-Rahmenwerks integriert. Abschließend lässt sich festhalten, dass die von uns gewünschte Vereinfachung der Werkzeugkonstruktion und die einfachere Wiederverwendbarkeit erreicht werden konnte.

6.2 Ausblick

Diese ist nach unserer Auffassung ein Schritt in Richtung einer ökonomischeren Anwendungsentwicklung. Der Softwareentwicklungsprozess kann durch Teilautomatisierungen von routineartigen Entwicklungsarbeiten, vor allem durch geeignete Werkzeuge, effizienter gestaltet werden. Ein Entwicklungswerkzeug, das die vorhandenen Werkzeugkomponenten visuell darstellen kann, ist ein mögliches Werkzeug zur Effizienzsteigerung. Ein Anwendungsentwickler kann mit Hilfe eines solchen Entwicklungswerkzeuges seine Anwendung zum Teil durch komponieren der graphischen Repräsentationen der Werkzeugkomponenten erstellen. Das Entwicklungswerkzeug kann den Anwender auf der einen Seite bei der Auswahl und dem Auffinden der Werkzeugkomponenten und passender Materialien unterstützen, andererseits beispielsweise durch automatische Codegenerierung Routinearbeiten abnehmen. Eine visuelle Verknüpfung zweier Werkzeugkomponenten könnte man sich folgendermaßen vorstellen. In der Werkzeugkomponentenbeschreibung könnte die Information über die erforderliche Funktionalität, die eine Werkzeugkomponente von anderen Werkzeugkomponenten erwartet (Call-Out-Schnittstellen), hinterlegt werden. Die Werkzeugkomponentenbeschreibungs-Klasse könnte eine Liste von Typen zurückliefern. Das visuelle Entwicklungswerkzeug für Werkzeugkomponenten könnte mit Hilfe dieser Liste alle für die Nutzung der Werkzeugkomponente benötigen Dienste anzeigen. Der Anwendungsentwickler würde aus dem Pool der Werkzeugkomponenten eine entsprechende auswählen und die Komponenten miteinander verknüpfen.

Eine weitere Frage, die untersucht werden kann ist, inwieweit sich das WAM-Leitbild für eigenverantwortliche Expertentätigkeit auf den Arbeitsbereich eines Softwareentwicklers anwenden lässt. Die Erkenntnisse aus der Anwendungsentwicklung mit den WAM-Enwurfsmustern könnten am eigenen Arbeitsplatz reflektiert und überprüft werden. Werkzeugkomponenten könnten beispielsweise als Materialien auf einem speziellen Entwickler-Desktop dargestellt und mit entsprechenden Werkzeugen und Automaten bearbeitet werden.

Abbildungsverzeichnis

Literaturverzeichnis

[Bäumer98] D. Bäumer: *Softwarearchitekturen für die rahmenwerkbasierte Konstruktion großer Anwendungssysteme.* Dissertationschrift zur Vorlage an der Universität Hamburg, 1998.

[Büchi & Weck 97] M. Büchi, W. Weck: *A Plea for Grey-Box Components.* Turcu Centre for Computer Science, TUCS Technical Report No 122, 1997.

[Fröse99] F. Fröse: *Komponentenorientierte Werkzeugkonstruktion – Entwurf und Implementierung eines Werkzeugkomponentenmodells.* Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik, 1999.

[Gamma96] E. Gamma, R. Helm, R. Johnson, J.Vlissides: *Entwurfsmuster – Elemente wiederverwendbarer objektorientierter Software.* Übersetzung von D. Riehle, Bonn: Addison Wesley, 1996.

[Griffel98] F. Griffel: *Componentware – Konzepte und Techniken eines Softwareparadigmas.* Heidelberg: dpunkt-Verlag, 1998.

[JWAM2000] Apcon Workplace Solutions: *JWAM 1.4 Dokumentation.* http://www.jwam.de, 2000.

[Lippert99] M. Lippert: *Die Desktop-Metapher in Systemen nach dem Werkzeug- und Material-Ansatz.* Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik, 1999.

[Meyer97] B. Meyer: *Object-Oriented Software Construction*. New York, London: Prentice-Hall, Second Edition, 1997.

[Pree97] W. Pree: *Komponentenbasierte Softwareentwicklung mit Frameworks*. Heidelberg: dpunkt-Verlag, 1997.

[Riehle98] D. Riehle: *Pattern Languages of Program Design 3*. Herausgeber: R. Martin, D. Riehle, F. Buschmann, Massachusetts: Addison-Wesley, 1998.

[Roock & Wolf 98] S. Roock, H. Wolf: *Die Raummetapher zur Entwicklung kooperationsunterstützender Softwaresysteme für Organisationen.* Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik, 1998.

[Sun97] Sun Microsystems: *JavaBeans API Specification.* http://java.sun.com/beans, 1997.

[Szyperski98] C.Szyperski: *Component Software – Beyond Object-Oriented Programming.* New York: Addison Wesley, 1998.

[Züllighoven98] H.Züllighoven: *Das objektorientierte Konstruktionshandbuch nach dem Werkzeug & Material-Ansatz.* Heidelberg: dpunkt-Verlag, 1998.