Diplomarbeit

Zugriffskontrollen für WAM-Systeme

Christian Beis
Im Wiesengrund 6
25488 Holm
Matrikel-Nr. 4725398
4beis@informatik.uni-hamburg.de

November 2000

Erstbetreuung: Prof. Dr. Heinz Züllighoven
Zweitbetreuung: Prof. Dr. Klaus Brunnstein

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

Erklärung:

Hiermit versichere ich, diese Arbeit selbständig und unter ausschließlicher Zuhilfenahme der in der Arbeit aufgeführten Quellen und Hilfsmittel erstellt zu haben.

Holm, den

Christian Beis
Im Wiesengrund 6
25488 Holm
Matrikel-Nr. 4725398

Betreuung:

Prof. Dr. Heinz Züllighoven (Erstbetreuer)
Prof. Dr. Klaus Brunnstein (Zweitbetreuer)

Prof. Dr. Heinz Züllighoven
Arbeitsbereich Softwaretechnik (SWT)
Fachbereich Informatik
Universität Hamburg
Vogt-Kölln-Straße 30
22527 Hamburg

Prof. Dr. Klaus Brunnstein
Arbeitsbereich Anwendungen der Informatik in Geistes- und Naturwissenschaft (AGN)
Fachbereich Informatik
Universität Hamburg
Vogt-Kölln-Straße 30
22527 Hamburg

Danksagungen

An dieser Stelle möchte ich allen denen bedanken, die mir bei der Erstellung dieser Arbeit auf die eine oder andere Art und Weise behilflich waren. Prof. Dr. Heinz Züllighoven und Prof. Dr. Klaus Brunnstein danke ich für die Betreuung der Arbeit sowie für viele hilfreiche Anregungen und Hinweise. Stefan Roock danke ich für viele fruchtbare Diskussionen und hilfreiche Anregungen. Ebenso bedanken möchte ich mich bei den Mitgliedern der JWAM-Architekturgruppe für Anregungen und Hinweise in Bezug auf die Konstruktion.

Meinen Eltern, meinem Bruder sowie Corinna Schmelzle danke ich für das Korrekturlesen dieser Arbeit.

Inhaltsverzeichnis

Kapitel 1 - Einleitung

1.1. Motivation

Der WAM-Ansatz (siehe [Züllighoven et al. 98]) bietet durch seine Leitbilder und Metaphern eine „prägnante Anleitung zur Konstruktion anwendungsorientierter Software“[1], wobei sich anwendungsorientierte Software durch eine „aufgabengerechte Funktionalität“, „benutzergerechte Handhabung und Präsentation“ und „Anpassungsfähigkeit“ an eine „Anwendungssituation“ ([Züllighoven et al. 98]) auszeichnet. Das am häufigsten verwendete Leitbild ist der „Arbeitsplatz für eigenverantwortliche Expertentätigkeit“. Das Anwendungssystem hat dabei die Aufgabe, den Anwender bei seiner täglichen Arbeit zu unterstützen. Bei der Modellierung eines Anwendungssystems für einen solchen Arbeitsplatz werden die Entwurfsmetaphern *Arbeitsumgebung*, *Werkzeug*, *Material*, *Behälter* und *Automat* verwendet[2]. Die zur Erledigung einer Aufgabe notwendigen Werkzeuge und Materialien werden dem Benutzer innerhalb seiner Arbeitsumgebung zugänglich gemacht.

Sobald mehrere Anwender zusammenarbeiten, existieren häufig Konventionen und Regeln, wie bestimmte Benutzer oder Mengen von Benutzern mit Werkzeugen und Materialien umgehen sollen. Meist ist die Benutzbarkeit von Werkzeugen und Materialien für Anwender an Zugriffsberechtigungen geknüpft. Zugriffsschutz bedeutet in diesem Sinne, daß nur autorisierte Anwender geschützte fachliche Gegenstände benutzen können. Möchte ein Anwender ein Werkzeug verwenden, wird überprüft, ob der Anwender das Recht zur Benutzung dieses Werkzeugs hat: entweder wird der Anwender bei jeder Benutzung des Werkzeugs aufgefordert, seine Zugriffsberechtigung nachzuweisen (wie z.B. an einem Geldautomaten durch Eingabe einer PIN), oder er meldet sich einmal beim System an und kann – solange er angemeldet ist – alle Systemfunktionen nutzen, für die er eine Zugriffsberechtigung hat. In diesem Fall benötigt das Anwendungssystem neben dem Wissen, welcher Anwender was darf, auch ein fachliches Modell des gerade angemeldeten Anwenders.

In einer verteilten oder kooperativen Arbeitsumgebung werden die fachlichen Modelle von Zugriffsberechtigungen und Anwendern zu einem wesentlichen Bestandteil des Anwendungssystems. Diese Modelle sollen zum einen hinreichende Sicherheit vor nicht autorisierter Benutzung von Werkzeugen und Materialien bieten, zum anderen müssen sie für die Anwender des Systems einfach zu verstehen und zu handhaben sein. Um dies zu erreichen, müssen Sicherheitskonzepte in Zusammenarbeit mit den Anwendern erarbeitet werden, wie A. Adams und M. Sasse in [Adams, Sasse 99] fordern:

> „System security is one of the last areas in IT in which user-centred design and user training are not regarded as essential – this has to change.”

Die Zusammenarbeit von Anwendungsentwicklern und Anwendern ist ein zentraler Aspekt des WAM-Ansatzes. Die genannten Entwurfsmetaphern sollen Anwendern und Entwicklern helfen, ein gemeinsames Verständnis für das zu konstruierende Anwendungssystem zu entwickeln.

Über die Modellierung des Anwenders und seiner Zugriffsrechte werden im Rahmen des WAM-Ansatzes bisher keine detaillierten Aussagen gemacht. In [Roock & Wolf 98] wird lediglich vorgeschlagen, eine im Rahmen des WAM-Ansatzes neue Metapher, die Schlüssel-Schloß-Metapher, einzuführen. Das dort vorgestellte Konzept beschreibt die Metapher aber nur oberflächlich. Erstmals wird eine Zugriffskontrolle für IT-Systeme mit Schlüsseln und Schlössern 1984 in „Access Control with Single-Key-Lock“ ([Wu, Hwang 84]) beschrieben. Allerdings wird die Metapher dort in gänzlich anderer Weise verwendet, denn in dem vorge-

[1] Zitat von Erich Gamma auf einer Veranstaltung im Rahmen eines Kolloquiums-Vortrags am Fachbereich Informatik der Universität Hamburg, Januar 1999

[2] Die Entwurfsmetaphern *W*erkzeug, *A*utomat und *M*aterial haben dem WAM-Ansatz seinen Namen gegeben.

schlagenen Modell läßt sich die Schlüssel-Schloß Metapher nur auf mathematisch- formaler Ebene erkennen. Die Zugriffskontrolle soll für Anwender und Anwendungsentwickler aber möglichst intuitiv verständlich und einfach zu handhaben sein.

Damit die Schlüssel-Schloß-Metapher im Rahmen des WAM-Ansatzes verwendbar ist, muß sie neu definiert werden. Dabei ist insbesondere die „Gegenständlichkeit“ von Schlüsseln und Schlössern sowie der Bezug zu den anderen Metaphern des WAM-Ansatzes zu berücksichtigen. Auf Basis dieser Betrachtungen kann dann ein Zugriffskontrollmechanismus für den WAM-Ansatz herausgearbeitet werden. Dieser Mechanismus wird Anwendungsentwicklern die Möglichkeit geben, die von ihnen modellierten fachlichen Gegenstände des Anwendungsbereichs vor nicht autorisiertem Gebrauch zu schützen.

Als Konsequenz der Modellierung von Zugriffsrechten und Benutzern wird sich die Handhabung von WAM-Systemen verändern, wobei sie sich weiterhin im Rahmen der Metaphern bewegen soll. Beispielsweise wird es erforderlich sein, einen Behälter aufzuschließen, bevor ein Material hinein gelegt werden kann.

1.2. Ausgangspunkte der Arbeit

Methodischer Rahmen

Der methodische Rahmen dieser Arbeit ist die Softwareentwicklungsmethode WAM. Der WAM-Ansatz unterstützt Anwendungsentwickler beim Entwurf objektorientierter Anwendungssysteme. Diese Arbeit enthält keine zusammenfassende Beschreibung des WAM-Ansatzes, da eine solche Beschreibung bereits Bestandteil diverser Publikationen ist (siehe z.B. [Züllighoven et al. 98], Seite 4ff oder [Bleek 97], Seite 9-15). Statt dessen werden Begriffe oder Konzepte aus dem WAM-Ansatz eingeführt, wenn sie im Rahmen dieser Arbeit relevant sind. Der konstruktive Anteil dieser Arbeit basiert auf dem JWAM-Rahmenwerk, welches im folgenden vorgestellt wird.

Das JWAM-Rahmenwerk

Größere Anwendungssysteme lassen sich heutzutage kaum noch effizient komplett neu entwickeln. Vielmehr ist die Wiederverwendung bereits existierender Lösungen und Lösungsansätze gefragt. Daher wurde am Arbeitsbereich Softwaretechnik der Universität Hamburg in Zusammenarbeit mit der APCON WPS (Hamburg) das JWAM-Rahmenwerk (Java-Rahmenwerk für WAM) gebaut. Es bietet für den Bereich „interaktiver Anwendungsssoftware für kooperative Arbeitsumgebungen“ „Ansatzstücke“ und eine „Anleitung für die technische Realisierung“ an ([Gryczan et al. 99a]). Die Anleitung wird dabei in Form des Leitbildes und der Entwurfsmetaphern gegeben. Diese helfen dem Anwendungsentwickler, eine bestimmte Sichtweise auf das Anwendungssystem einzunehmen und unterstützen ihn bei Entwurf und Konstruktion.

Der Kern des Rahmenwerks bietet grundlegende Unterstützung zur Werkzeug- und Materialkonstruktion an. Hinzu kommen Erweiterungen, die auf dem Kern basieren. Beispiele sind Komponenten wie der JWAM-Desktop (beschrieben in [Lippert 99]) oder eine fachliche Registratur (siehe [Havenstein 99]). Der Anwendungsentwickler kann auf diesem Kern aufbauen und bestehende Komponenten anpassen oder eigene Komponenten entwickeln und integrieren. Die im Kern vorhandene Unterstützung zur Werkzeug- und Materialkonstruktion und die genannten Erweiterungen ermöglichen die Konstruktion von softwareunterstützten Einzelarbeitsplätzen und schaffen eine Voraussetzung für kooperative Arbeitsumgebungen.

1.3. Ziel der Arbeit

Diese Arbeit beschreibt ein fachliches Modell des Anwenders und seiner Zugriffsberechtigungen für WAM-Systeme. Es wird herausgearbeitet, warum sich die Schlüssel-Schloß-Metapher prinzipiell als Zugriffskontrollmodell für den WAM-Ansatz eignet. Aufbauend darauf wird die Schlüssel-Schloß-Metapher für den WAM-Ansatz definiert und ihr Einfluß auf die anderen Entwurfsmetaphern auf konzeptioneller Ebene dargestellt.

Für das JWAM-Rahmenwerk wird eine softwaretechnische Lösung für die erarbeiteten Konzepte entworfen und konstruktiv umgesetzt. Auf Basis dieser Konstruktion soll es Anwendungsentwicklern, die das JWAM-Rahmenwerk benutzen, auf einfache Weise möglich sein, Zugriffskontrollen in ein Anwendungssystem zu integrieren.

1.4. Aufbau der Arbeit

Kapitel 2 stellt vor, welche Sicherheitsanforderungen für WAM-Systeme in Bezug auf Zugriffskontrollen gelten. Hierzu werden Sicherheitsanforderungen im allgemeinen sowie die Begriffe *Zugriffskontrolle* und *Zugangskontrolle* im speziellen eingeführt. Aufbauend auf der Vorstellung des WAM-Ansatzes sowie seiner Einordnung in die Programm-Klassifikation nach [Lehman 80] werden spezielle Anforderungen für Zugriffskontrollen in WAM-Systemen herausgearbeitet und konventionelle Zugriffskontrollmodelle anhand dieser Anforderungen bewertet. Vor dem Hintergrund dieser Betrachtungen wird die Schlüssel-Schloß-Metapher als Zugriffskontrollmodell für den WAM-Ansatz vorgeschlagen.

Kapitel 3 beschreibt, wie sich ein fachliches Benutzermodell und die Schlüssel-Schloß-Metapher für den WAM-Ansatz definieren lassen. Es wird diskutiert, wie sich die Einführung der Schlüssel-Schloß-Metapher auf die WAM-Entwurfsmetaphern auswirkt und welche zusätzlichen Anforderungen sich aus der Berücksichtigung verschiedener Leitbilder sowie der Berücksichtigung kooperativer Arbeitssituationen ergeben.

Kapitel 4 stellt wesentliche Konzepte des JWAM-Rahmenwerkes vor und schildert, wie sich die zuvor auf fachlicher Ebene diskutierten Konzepte in einen softwaretechnischen Entwurf für das Rahmenwerk umsetzten lassen. Dabei werden die speziellen Anforderungen des fachlichen Benutzermodells zusammengetragen, ein Entwurf zur Modellierung von Schlüsseln und Schlössern vorgestellt und untersucht, wie sich Gegenstände im softwaretechnischen Modell mit Schlössern absichern lassen. Bei der Umsetzung in den Entwurf ergeben sich Veränderungen und neue Aspekte gegenüber dem ursprünglichen Benutzermodell bzw. gegenüber der ursprünglichen Metapher. Diese werfen zum Teil neue Fragestellungen auf, für die entsprechende Lösungsansätze vorgestellt werden.

Kapitel 5 zeigt, wie die Schlüssel-Schloß-Metapher im Rahmen des JWAM-Rahmenwerkes konstruktiv umgesetzt wurde. Im Detail werden außerdem die Konstruktion verschließbarer Gegenstände sowie das Abschließen von Werkzeugen, Subwerkzeugen und Behältern betrachtet.

Kapitel 6 faßt wesentliche Ergebnisse dieser Arbeit zusammen und stellt heraus, wie die im Laufe der Arbeit aufgetretenen offenen Fragen weitergehend bearbeitet werden können.

1.5. Abgrenzung der Arbeit

Jedes Anwendungssystem wird vor dem Hintergrund des Spannungsfeldes „maximale Sicherheit“ – „gute Benutzbarkeit“ konstruiert. Bei WAM-Systemen handelt es sich um interaktive Anwendungssysteme, die meist nach dem Leitbild „Arbeitsplatz für eigenverantwortliche Expertentätigkeit“ entworfen werden. Sie legen daher den Schwerpunkt auf die Benutzbarkeit des Systems, während Sicherheit kein zentraler Aspekt des Ansatzes ist. Anwendungssysteme, die mit JWAM konstruiert werden, abstrahieren von den zugrundeliegenden Technologien (Rechner- und Netzwerkarchitektur, Datenbanksysteme, Middleware usw.). In dieser Arbeit werden Mechanismen beschrieben, mit denen in WAM-Systemen der nicht autorisierte Zugriff auf softwaretechnische Gegenstände verhindert werden kann. Angriffe, die sich gegen die zugrundeliegenden Technologien bzw. die Programmiersprache Java richten, werden in dieser Arbeit nicht behandelt.

1.6. Konventionen

Werden in dieser Arbeit Begriffe eingeführt oder soll ein Begriff in einer speziellen Bedeutung verwendet werden, wird dies durch Kursivschrift kenntlich gemacht. Ergebnisse, Definitionen, sowie offene Fragen, die im Rahmen dieser Arbeit nicht beantwortet werden können, werden durch ein spezielles Absatzformat hervorgehoben.

Zur Darstellung statischer Beziehungen zwischen Klassen bzw. dynamischer Beziehungen zwischen Objekten werden Klassendiagramme bzw. Interaktionsdiagramme verwendet, wie sie die *Unified Modelling Language* (UML) in [Booch et al. 99] beschreibt. Elemente der Konstruktion werden im Text durch die Schriftart `Courier New` hervorgehoben. Für die Wahl der Bezeichner in der konkreten Konstruktion gilt der *styleguide* des JWAM-Rahmenwerkes. Dieser sieht unter anderem vor, daß im Quellcode englischsprachige Bezeichner gewählt werden.

Diese Arbeit ist in der *Übergangszeit zur Neuregelung der deutschen Rechtschreibung* entstanden. Die neuen Regelungen der Rechtschreibung werden in dieser Arbeit noch nicht berücksichtigt.

Als Zugeständnis an die Lesegewohnheiten werden im folgenden meist maskuline Bezeichnungen verwendet. Frauen sind selbstverständlich ebenfalls gemeint.

Kapitel 2 - Sicherheitsanforderungen für WAM-Systeme

Wie schon in der Einleitung erwähnt, bietet der WAM-Ansatz dem Anwendungsentwickler Unterstützung zur Konstruktion von Anwendungssystemen. Der Begriff *Anwendungssystem* wird im WAM-Kontext wie folgt definiert (nach ([Bleek 97], Seite 71):

> **Definition 1: Anwendungssystem**
> Unter einem Anwendungssystem wird die fachliche Zusammenfassung von Softwarewerkzeugen, Automaten und Arbeitsplatzsystemen zu einem Ganzen verstanden.

Allerdings gibt es bisher im WAM-Ansatz keine Aussagen darüber, wie Zugriffskontrollen bei Entwurf und Konstruktion berücksichtigt werden können. In diesem Kapitel wird vorgestellt, welche Modelle für Zugriffskontrollen es gibt, und wie diese im Rahmen des WAM-Ansatzes verwendbar sind.

2.1. Sicherheit und Sicherheitsanforderungen

Sicherheit ist in der Informationstechnologie (IT) heutzutage ein viel diskutiertes und in der Praxis oft vernachlässigtes Thema. Sicherheit ist erst einmal ein abstraktes Gut, mit dem sich nicht handeln läßt. Entsprechend sind die Ansätze zur Sicherung von IT-Systemen nicht aus dem kommerziellen, sondern dem militärischen Bereich heraus entstanden (siehe [Schier 99], S. 27). Sicherheit ist jedoch kein Zustand, sondern ein „immerwährender Prozeß hinsichtlich der Durchsetzung von Schutzzielen“ ([BSI 97]).

Die Integration von Sicherheitsmaßnahmen in ein Anwendungssystem bedeutet nicht nur bei Entwurf und Implementation einen Mehraufwand an finanziellen und personellen Mitteln. Auch beim Betrieb des Systems müssen Ressourcen für Unterhalt und Ausbau sowie für die Schulung der Mitarbeiter bereitgestellt werden. Dabei kann nicht einmal garantiert werden, daß die getroffenen Sicherheitsmaßnahmen ausreichend sind: die Sicherheit eines Systems ist nicht beweisbar, weil Sicherheit eine rein subjektiv wahrnehmbare Größe und „empirisch nicht verifizierbar“ ist ([Schier 99], S. 125).

2.1.1. Terminologie

In Hinblick auf IT-Systeme wird unter Sicherheit sowohl Ablauf- bzw. Ausfallsicherheit (engl. *safety*) als auch Schutz vor beabsichtigten Angriffen (engl. *security*) verstanden (siehe [Oppliger 97]). Obwohl das Ausmaß dieses Schutzes weder quantitativ noch qualitativ bestimmt werden kann, möchte man Aussagen über Sicherheitseigenschaften eines Systems machen. Zumindest in Bezug auf diese Eigenschaften kann das korrekte Verhalten des Systems verifiziert werden. Einige dieser Eigenschaften sind Vertraulichkeit (engl. *confidentiality*), Integrität (engl. *integrity*) und Verfügbarkeit (engl. *availability*):

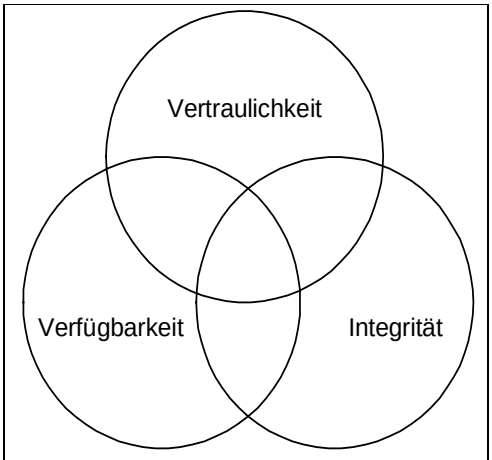


Abbildung 1: Vertraulichkeit, Integrität und Verfügbarkeit Quelle: [Pfleeger 1997], S. 5

> **Definition 2: Vertraulichkeit**
> Schutz vor unbefugter und unbeabsichtigter Kenntnisnahme von Informationen. (nach [ITSEC 91])

> **Definition 3: Integrität**
> Schutz vor unbefugter und unbeabsichtigter Veränderung von Informationen. (nach [ITSEC 1991])

Definition 4: Verfügbarkeit
Schutz vor unbefugter und unbeabsichtigter Vorenthaltung von Informationen oder Betriebsmitteln. (nach [ITSEC 1991])

Wie Abbildung 1 zeigt, sind diese Anforderungen größtenteils unabhängig voneinander und überschneiden sich nur teilweise. Die genannten drei Eigenschaften werden in [Schier 99] auch als „traditionelle“ Sicherheitsanforderungen bezeichnet. Sie stammen im wesentlichen aus dem Bereich der militärischen Datenverarbeitung. Ihre Zielsetzung ist, daß ein berechtigter Anwender zu jedem Zeitpunkt korrekte Daten aus einem Informationssystem erhalten kann und ein unberechtigter Anwender nicht einmal die Existenz dieser Daten erfährt. Darüber hinaus gibt es weitere Sicherheitsanforderungen, die den Benutzer des Systems mehr in das Zentrum der Betrachtung rücken:

Definition 5: Zurechenbarkeit (engl. *accountability*)
Eine Handlung ist einer Person zurechenbar, diese kann für die Handlung verantwortlich gemacht werden. (nach [Schier 99], S. 31)

Definition 6: einfache Handhabung (engl. *ease of use*)
Die Benutzung von Anwendungssystemen muß so gestaltet sein, daß sie dem Benutzer leicht fällt und seiner Intuition entspricht. Sicherheitsmaßnahmen dürfen keinen zusätzlichen Aufwand in der Benutzung darstellen, wenn sie einen tatsächlichen Sicherheitsvorteil bieten sollen. (siehe [Schier 99], S. 34)

Die einfache Handhabbarkeit von Sicherheitsmaßnahmen ist ein wesentlicher und in der Vergangenheit häufig vernachlässigter Aspekt für die Gestaltung eines Anwendungssystems. Anwender neigen dazu, die Sicherheitsmechanismen eines Anwendungssystems bewußt zu umgehen, wenn sie deren Sinn und Funktion nicht verstehen (vgl. [Adams, Sasse 99]).

Weiterhin existieren noch verschiedene Sicherheitsanforderungen, die im wesentlichen auf den Schutz der über eine Person gesammelten Daten zielen (z.B. das „Recht auf informationelle Selbstbestimmung“) . Diese besitzen eine hohe Relevanz bei der Entwicklung konkreter Anwendungssysteme (zum Beispiel eines Krankenhausinformationssystems). Dabei sind die jeweiligen Datenschutzgesetze der Bundesländer sowie das Bundesdatenschutzgesetz ([BDSG]) zu beachten. Diese Betrachtungen stehen aber nicht im Zentrum dieser Arbeit und werden daher nicht weiter behandelt. Mehr hierzu ist z.B. in [Schier 99] (S. 32ff) zu finden.

2.1.2. Sicherheit im Anwendungssystem

Offensichtlich gibt es eine Reihe von Sicherheitsanforderungen für Anwendungssysteme. Im konkreten Fall sind diese Anforderungen unterschiedlich relevant und werden in den Systementwurf entsprechend einbezogen. Berücksichtigt werden muß dabei neben der softwaretechnischen Architektur auch die technische Umgebung des Anwendungssystems.

Die softwaretechnische Architektur ist zunächst einmal für das problemlose Ablaufen der Anwendung auf einem Rechner verantwortlich. Grundprinzipien wie zum Beispiel Modularität (siehe [Züllighoven et al. 98], Seite 44ff) können die Qualität (und damit die Benutzbarkeit und Verfügbarkeit) sowie die Vertrauenswürdigkeit von Software deutlich erhöhen (vgl. [Pfleeger 97], S. 207ff). Darüber hinaus werden auf softwaretechnischer Ebene fachliche Sicherheitsanforderungen modelliert. Ein Beispiel ist eine Zugriffskontrolle für Gehaltsabrechnungen: auch ohne die Existenz eines Anwendungssystems gibt es in der Anwendungswelt Regeln, wer Gehaltsabrechnungen sehen oder bearbeiten darf. Diese Regeln werden im fachlichen Entwurf berücksichtigt und in das Anwendungssystem übernommen. Offenbar ist Sicherheit kein Selbstzweck, sondern wird immer durch fachliche Anforderungen motiviert.

Die Sicherheit, die die softwaretechnische Architektur erbringen soll, muß auf technischer Ebene unterstützt werden. Die bloße Verwendung fachlicher Zugriffskontrollen ist sinnlos, wenn Daten in eine nicht adäquat gesicherte technische Umgebung weitergeleitet und ge-

speichert werden. Die technische Umgebung umfaßt neben der verwendeten Hardware auch Softwarekomponenten (z.B. eine Datenbank oder ein Betriebssystem). In Bezug auf die Sicherheit eines Anwendungssystems ist die technische Umgebung von großer Bedeutung, da hier Sicherheitsprobleme auftreten können, die nicht im Einflußbereich der Softwareentwickler liegen. Eklatante Sicherheitsschwächen in einer technischen Komponente machen sämtliche Bemühungen um Sicherheit auf softwaretechnischer Ebene zunichte. Ein Beispiel für Sicherheit auf technischer Ebene ist die Verwendung sicherer Übertragungsprotokolle in verteilten Anwendungssystemen.

2.2. Zugangskontrollen und Zugriffskontrollen

Im weiteren Verlauf dieser Arbeit wird mit *Zugriffskontrollen* ein spezieller Sicherheitsmechanismus betrachtet. Zugriffskontrollen zielen primär auf den Schutz der Vertraulichkeit von Informationen ab. Betrachtet man ein Anwendungssystem als einfachen Prozeß, der auf einem Rechner abläuft, finden Zugriffskontrollen häufig im Betriebssystemkern oder im Dateisystem statt, ohne daß dies in der Anwendung selbst implementiert werden muß.

Im Rahmen dieser Arbeit werden Zugriffskontrollen auf dieser Ebene nicht betrachtet, da sie in der technischen Umgebung (siehe Abschnitt 2.1.2) angesiedelt sind. Hier sollen nur Zugriffskontrollen betrachtet werden, die aus den fachlichen Anforderungen des Anwendungsbereiches motiviert sind. Die Konzepte dieser Zugriffskontrollen werden daraufhin untersucht, ob sie sich auch für den Schutz anwendungsfachlicher Gegenstände eignen.

2.2.1. Zugangskontrollen

Bei den meisten Anwendungssystemen ist es notwendig, dem System mitzuteilen, welcher Anwender mit dem System arbeitet. Der Vorgang, bei dem ein Benutzer in Zusammenhang mit einer im System bekannten Identität in Zusammenhang gebracht wird, heißt *Zugangskontrolle*. Wird diese Identität auf ihre Echtheit überprüft, spricht man von Authentifizierung (auch: Authentisierung, siehe [Rankl, Effing 1996], S. 268).

> **Definition 7: Zugangskontrolle**
> Die Zugangskontrolle hat die Identität eines Benutzers festzustellen und zu entscheiden, ob dieser Zugang erhält. Nach erfolgter Zugangskontrolle gilt der Anwender im System als angemeldet und der Anwender kann das System entsprechend seinen Zugriffsrechten benutzen. (nach [Oppliger 97], Seite 173)

> **Definition 8: Authentifizierung**
> Unter Authentifizierung versteht man die Verifikation einer zuvor bekannten Identität. (nach [Oppliger 97], Seite 173)

Bei vielen interaktiven Anwendungssystemen wird nach der Authentifizierung eine persönliche Arbeitsumgebung (z.B. persönlicher Desktop) geladen und präsentiert. Alle Aktionen, die von dem jeweiligen Arbeitsplatz aus unternommen werden, können nun diesem Benutzer zugeordnet werden.

Juristisch gesehen handelt es sich bei solchen Aktionen dann um eine sogenannte *Willenserklärung* (siehe [Schier 99], Seite 117). Eine Willenserklärung ist die „Äußerung eines rechtlich bedeutsamen Willens“ ([Deinert, Hövel 00]). Auf die juristische Bedeutung dieses Begriffes wird im Rahmen dieser Arbeit nicht weiter eingegangen[3]. Bezogen auf Zugangskontrollen bedeutet die Willenserklärung, daß der Anwender die Verantwortung für alle von ihm vorgenommenen Aktionen übernimmt. Daher muß ihm bewußt sein, wann er eine Zugangskontrolle passiert hat. Gängige Verfahren zur Authentifizierung sind (siehe [Rankl, Effing 1996], S. 258ff):

[3] Näheres hierzu siehe ([BGB], § 116 ff).

- Der *besitzbasierte* Ansatz: der Anwender ist im Besitz eines Gegenstandes (z.B. einer Chipkarte). Wenn er diesen Gegenstand präsentiert, ist seine Identität bewiesen.
- Der *wissensbasierte* Ansatz: der Anwender authentifiziert sich durch Eingabe eines Geheimnisses (z.B. Paßworte oder PIN[4]), welches nur ihm bekannt sein sollte.
- *Biometrischen Verfahren* identifizieren einen Anwender aufgrund individueller Merkmale seiner Person (z.B. Fingerabdruck oder Unterschrift)[5]. Da davon ausgegangen wird, daß diese Merkmale nur einer Person zuzuordnen sind, ist die Identifizierung gleichzeitig eine Authentifizierung.

Alle drei genannten Verfahren haben alleine angewandt ihre Schwächen. Gegenstände können gestohlen oder verliehen werden und Geheimcodes können erpreßt, verraten, erraten oder einfach weitergegeben werden. In Bezug auf Paßwörter liegen diverse Untersuchungen vor, die auf Probleme bei der Verwendung hinweisen[6]. Dabei überwiegt nicht das Ausspähen oder Erpressen von Paßwörtern, sondern vielmehr die Wahl von und der Umgang mit Paßwörtern. Anwender verwenden vielfach leicht zu erratende Paßwörter (z.B. Vornamen, Geburtsdatum), geben sie an Kollegen weiter oder machen sich Notizen, die an einer leicht zugreifbaren Stelle am Arbeitsplatz zu finden sind.

Die in Bezug auf die Fehlerrate sehr sicheren biometrischen Verfahren sind umstritten, weil hier persönliche Daten gespeichert werden müssen. Dies kann zu datenschutzrechtlichen Problemen führen. Biometrische Zugangskontrollen können auch ohne eine Handlung des Anwenders erfolgen. Damit ist dem Anwender nicht mehr unbedingt bewußt, daß er eine Willenserklärung in Bezug auf die Benutzung eines Anwendungssystems abgibt.

Letztlich ist es Aufgabe der Entwickler eines konkreten Anwendungssystems, eine geeignete Zugangskontrolle auszuwählen. Oft wird die Zugangskontrolle des Arbeitsplatz-Betriebssystems genutzt und als ausreichend empfunden, so daß sich der Anwender im Endeffekt nur einmal anmelden muß (*Single Login*, auch gebräuchlich: *Single Sign In*).

2.2.2. Zugriffskontrollen

Nach erfolgter Zugangskontrolle kann der Anwender mit dem Anwendungssystem arbeiten. Sobald er eine Aktion auf einem Objekt im Anwendungssystem ausführt (z.B. das Anzeigen oder Öffnen einer Datei), findet eine *Zugriffskontrolle* statt:

> **Definition 9: Zugriffskontrolle**
> Die Zugriffskontrolle hat für einen authentifizierten Benutzer zu entscheiden, ob dieser Zugriff auf ein bestimmtes Objekt[7] erhält. (nach [Oppliger 97], Seite 173)

Man sagt auch, der Anwender sei autorisiert, die Aktion auf dem Objekt auszuführen. Die Zugriffskontrolle entscheidet dabei anhand vorher definierter Rechte, ob ein Zugriff erlaubt ist. Wie Zugriffskontrollen durchgeführt und Rechte vergeben werden, beschreibt ein Zugriffskontrollmodell. Dabei wird folgende Terminologie verwendet (siehe [Beis, Vorwerk 97] und [Schier 99]):

> **Definition 10: Anwender / Benutzer**
> Ein Anwender ist eine natürliche Person. Synonym wird der Begriff „Benutzer“ verwendet.

[4] PIN = *Personal Identification Number* - eine Zahl, die nur einem Anwender bekannt ist und mit der er sich authentifizieren kann.

[5] Einen Überblick über biometrische Verfahren geben [Henke 00] und [Jain et al. 00].

[6] Siehe z.B. [Oppliger 97] (S. 181ff), [Adams, Sasse 99], [Silberschatz 98] (S. 625ff), [Pfleeger 97] (S. 254ff)

[7] Unter *Objekten* werden in diesem Zusammenhang nicht Exemplare von Klassen (im Sinne der Objektorientierung) sondern Systemressourcen bzw. Betriebsmittel verstanden.

Definition 11: Subjekt
Subjekte sind die aktiven Einheiten eines Systems, zum Beispiel Benutzer oder Prozesse. Ein von einem Benutzer gestarteter Prozeß verfügt über die gleichen Zugriffsrechte wie der Benutzer.

Definition 12: Transaktion / Aktion
Transaktionen sind Handlungen/Prozeduren in einer bestimmten Zugriffsart, die auf Objekte angewendet werden. Synonym wird der Begriff „Aktion" verwendet.

Definition 13: Objekt
Auf Objekten dürfen autorisierte Transaktionen ausgeführt werden.

Generell lassen sich die verschiedenen Zugriffskontrollmodelle in diskretionäre (engl. *discretionary access control*, DAC), mandatorische (engl. *mandatory access control*, MAC) sowie rollenbasierte Ansätze (engl. *role based access control*, RBAC) einteilen.

Die diskretionäre Zugriffskontrolle basiert auf dem Eigentümer-Prinzip: jedes Objekt im Anwendungssystem hat einen Besitzer. Dieser bestimmt die Zugriffsrechte der anderen Subjekte auf das Objekt. Im Gegensatz zur diskretionären Zugriffskontrolle liegt die Vergabe von Zugriffsrechten bei mandatorischer Zugriffskontrolle nicht im Ermessen eines Objektbesitzers. Vielmehr erfolgt hier regelbasiert eine Kontrolle des Informationsflusses. Subjekte und Objekte werden vom Systemadministrator in Sicherheitsklassen eingeteilt und der Informationsfluß zwischen den Sicherheitsklassen geregelt. Bei rollenbasierten Ansätzen obliegt die Vergabe der Zugriffsrechte ebenfalls einem Administrator. Die Rechte werden dabei nicht Individuen, sondern Rollen zugewiesen.

Eine detailliertere Betrachtung findet sich im folgenden. Es werden exemplarisch einige Zugriffskontrollmodelle vorgestellt, die im wesentlichen die Vertraulichkeit sicherstellen sollen. Eine Betrachtung und Bewertung aller existierenden Zugriffskontrollmodelle würde den Rahmen dieser Arbeit sprengen. Einen Überblick über Zugriffskontroll- und Sicherheitsmodelle verschafft [Schier 99].

Diskretionäre Zugriffskontrollen mit Zugriffskontrollisten
In der Praxis wird das Prinzip der diskretionären Zugriffskontrolle häufig durch Zugriffskontrollisten umgesetzt. Aufgrund ihrer weiten Verbreitung sollen sie hier exemplarisch untersucht werden. Eine Zugriffskontrolliste (engl. *access control list, ACL*) ist einem Objekt zugeordnet. Ein Listeneintrag legt das Recht eines Anwenders fest, eine bestimmte Transaktion auszuführen. In Abbildung 2 ist ein Beispiel für eine Zugriffskontrolliste zu sehen.

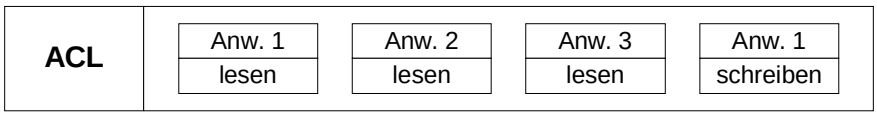


Abbildung 2: Eine Zugriffskontrolliste

Nach dieser Liste hätten die Anwender „Anw. 1", „Anw. 2" und „Anw. 3" Leserechte und „Anw. 1" zusätzlich Schreibrechte auf dem gesicherten Objekt.

Der Vorteil von Zugriffskontrollisten ist, daß Anwender individuell Objekte im Anwendungssystem schützen können: „Access Control Lists correspond directly to the needs of the users." ([Silberschatz 98], Seite 610). Sie sind einfach zu implementieren und für technisch versierte Anwender verständlich und bedienbar, weswegen sie in vielen kommerziellen Systemen verwendet werden. Beispiele sind Windows NT/Windows 2000 ([MSDN 99a]) und Unix ([Tanenbaum 95], Seite 241).

Zugriffskontrollisten können in großen Systemen sehr umständlich in der Handhabung werden ([Ferraiolo, Gilbert, Lynch 93], [Barkley 97]). Wesentliche Gründe hierfür sind:

- Mit Zugriffskontrollisten lassen sich die Strukturen größerer Organisationen nicht effizient nachbilden. Berechtigungen können nur für Benutzer oder Benutzergruppen gegeben werden, eine Orientierung an Aufgaben oder Rollen ist nicht möglich.
- Bei großen Mengen von Subjekten und Objekten muß unverhältnismäßig viel Aufwand durch Systemadministratoren betrieben werden, wenn sich Zugriffsberechtigungen ändern.
- Eine zeitnahe Veränderung von Zugriffsrechten für noch aktive Benutzer ist kaum durchführbar, da viele Zugriffskontrollisten in ständigem Gebrauch sind.

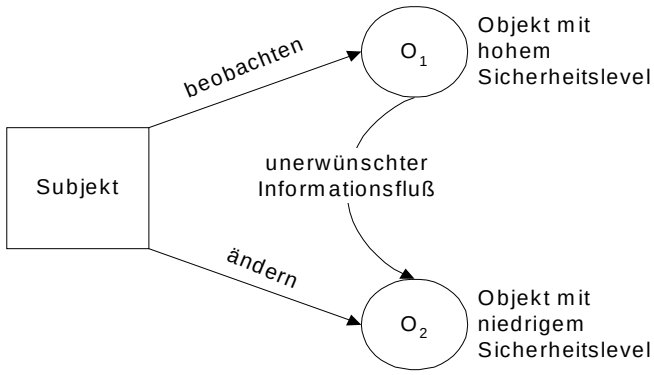


Abbildung 3: Unerlaubter Informationsfluß (nach [Kessler, Mund 93])

Ein generelles Problem bei diskretionärer Zugriffskontrolle ist, daß nicht der Informationsfluß (s. Abbildung 3), sondern nur der Datenzugriff kontrolliert wird. Kann zum Beispiel ein Subjekt ein Objekt O_1 mit hohem Sicherheitsniveau beobachten und gleichzeitig ein Objekt O_2 mit niedrigem Sicherheitsniveau ändern, so kann es Daten von dem höher klassifizierten zum niedrig klassifizierten Objekt kopieren (vgl. [Schier 99], Seite 132).

Es findet also ein Informationsfluß von O_1 nach O_2 statt, der durch das Zugriffskontrollsystem nicht verhindert werden kann, da die Zugriffe auf O_1 und O_2 erlaubt sind. Es würde aber auch keinen Sinn machen, dem Anwender den Zugriff auf unterschiedlich klassifizierte Objekte grundsätzlich zu verbieten.

Diskretionäre Zugriffskontrolle mit Single-Key-Locks (SKL)

Einen gänzlich anderen Ansatz verfolgen M.L. Wu und T.Y. Hwang in [Wu, Hwang 84]. Hier taucht erstmals eine Konkretisierung des Gedankens auf, Zugriffskontrollen mit Schlüsseln und Schlössern durchzuführen. Die Grundidee ist, jedes Objekt in einem System mit einem Schloß und jedes Subjekt mit einem Schlüssel zu versehen. Solche Systeme werden auch Fähigkeiten-basierte Systeme genannt, da im Unterschied zu Zugriffskontrollisten nicht die Rechte aller Anwender auf einem Objekt, sondern die Rechte eines Anwenders für alle Objekte beschrieben werden.

Schlüssel und Schlösser im SKL-Modell werden durch mathematische Funktionen berechnet. Hierzu existieren verschiedene Verfahren (u.a. [Wu, Hwang 84], [Chang, Jiang 89] und [Hwang, Shao, Wang 92]). Das in [Hwang, Shao, Wang 92] vorgestellte Verfahren wird hier näher erläutert. Es beruht auf dem Faktorisierungstheorem, welches besagt, daß sich jede Integerzahl $z > 1$ als eindeutige Folge von Primzahlen darstellen läßt: $z = p_1^{n1}\ p_2^{n2} \ldots p_r^{nr}$, wobei die p_i unterschiedliche Primzahlen sind ($p_1 < \ldots < p_r$) (siehe [Andreae 93], Seite 25). Beispielsweise kann die Zahl 180 zerlegt werden in $2^2\ 3^2\ 5^1$. Die Primzahl p_i ist dabei n_i-mal in z enthalten. In der folgenden Zugriffskontrollmatrix werden die Anwender (Subjekte S_1 bis S_4) zu den Objekten (O_1 bis O_6) in Beziehung gesetzt. Ein Eintrag r_{ij} beschreibt das Zugriffsrecht von Subjekt i auf Objekt j. Zugriffsrechte sind hierarchisch geordnet, wobei 0 das

	O_1	O_2	O_3	O_4	O_5	O_6
S_1	4	0	3	0	4	3
S_2	0	2	4	2	0	4
S_3	1	4	0	0	1	2
S_4	1	0	1	4	0	0

Abbildung 4: Zugriffskontrollmatrix (aus [Oppliger 97])

niedrigste und 4 das höchste Zugriffsrecht ($zugriffsRecht_{Max}$) ist. Den Subjekten werden als Schlüssel Primzahlen zugeordnet, also z.B. $K_1=2$, $K_2=3$, $K_3=5$, $K_4=7$ (siehe Abbildung 4). Ein Schloß L errechnet sich dann aus $L_j=\prod^4_{i=1} K_i^{rij}$. Es ergibt sich: $L_1 = \prod^4_{i=1} K_i^{ri1} = 2^4 3^0 5^1 7^1 = 560$, $L_2 = \prod^4_{i=1} K_i^{ri2} = 2^0 3^2 5^4 7^0 = 5625$ usw., wobei L_i das Schloß des Objektes O_i ist.

Will ein Subjekt auf ein Objekt zugreifen, können die Zugriffsrechte des Subjektes mit dem Algorithmus aus [Hwang, Shao, Wang 92] berechnet werden. Dieser Algorithmus basiert darauf, daß ein Schlüssel K genau z mal in der faktorisierten Darstellung von L enthalten ist (s.o.), wobei z das Zugriffsrecht ist.

Ein Vorteil dieses Modells ist, daß der Aufwand für die Einführung neuer Objekte sowie Änderung von Zugriffsrechten bei bestehenden Objekten sehr gering ist. Das entsprechende Schloß muß einfach nur neu berechnet werden. Dies erleichtert die Administrierbarkeit im Vergleich zu Zugriffskontrollisten erheblich.

In den bisher vorgestellten Varianten eines SKL-Systems erhalten Anwender jeweils nur genau einen Schlüssel und können diesen auch nicht weitergeben. Ebensowenig wird die Modellierung von Benutzergruppen unterstützt. Die Weitergabe einzelner Rechte ist allerdings möglich, wenn ein Subjekt über das Recht verfügt, für ein Objekt Rechte zu vergeben.

Mandatorische Zugriffskontrolle: das Vetraulichkeitsmodell nach Bell & LaPadula

Das erste formale, hierarchische mandatorische Zugriffskontrollmodell wurde zwischen 1973 und 1976 von D.E. Bell und L.J. LaPadula entwickelt und beschreibt, wie sich der Informationsfluß und damit die Vertraulichkeit sicherstellen lassen (siehe [Oppliger 97], S. 213ff). Vereinfacht dargestellt beruht dieses Modell darauf, daß Informationen von einem Element (Subjekt oder Objekt) immer nur zu gleich- oder höherwertig klassifizierten Elementen fließen dürfen. Zum Ausdruck gebracht wird dies durch zwei Regeln (vgl. Abbildung 5):

- Ein klassifiziertes Subjekt darf nur niedriger oder gleich klassifizierte Objekte betrachten („no read up“).
- Ein klassifiziertes Subjekt darf nur höher oder gleich klassifizierte Objekte ändern („no write down“).

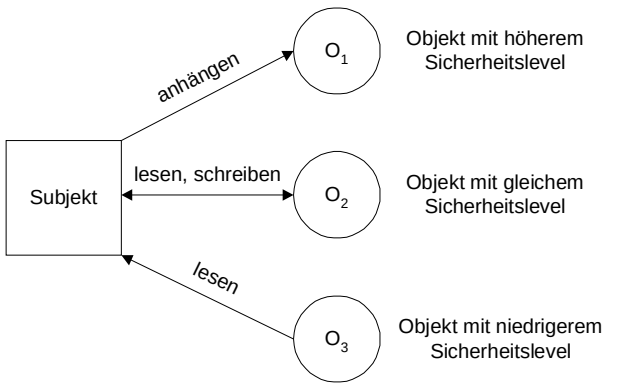


Abbildung 5: Informationsfluß im Bell-LaPadula Vertraulichkeits-Modell (nach [Kessler, Mund 93])

[Oppliger 97] äußert verschiedene Kritikpunkte am Bell-LaPadula-Modell (BLP-Modell). Die Anzahl von Zugriffsarten (z.B. Lesen, Anhängen, Schreiben, Ausführen) sei beschränkt und komplexere Transaktionen ließen sich daher nicht abbilden. Weiterhin könne man die Klassifikation von Objekten nur erhöhen, so daß Anwender mit einer niedrigeren Klassifikation höher klassifizierte Objekte in keinem Fall betrachten könnten. Dies entspricht mutmaßlich eher einer militärischen als einer kommerziellen Büroarbeits-Situation.

Rollenbasierte Ansätze

Mit rollenbasierten Zugriffskontrollen wurde ein weiteres Zugriffskontrollmodell entworfen, welches die flexible Gestaltung der Benutzerrechte erlaubt siehe (siehe z.B. [Ferraiolo, Kuhn 92]). Die Zugriffsrechte des Benutzers sind dabei nicht an ihn als Person, sondern als Mitglied einer *Rolle* gebunden. Der Begriff *Rolle* wird wie folgt definiert (vergleiche hierzu auch *funktionelle Rolle* in [Floyd 99]):

Definition 14: Rolle
Eine Rolle beschreibt Aufgaben, Verantwortung und Qualifikationen von Benutzern, ohne sich auf konkrete Personen zu beziehen.

Eine Definition für ein formales Rollenmodell findet sich in [Schier 99] (Seite 155). Rollen werden verschiedene Transaktionen zugeordnet (siehe Abbildung 6), mit denen Anwender Aufgaben erledigen, die aus ihrer Zugehörigkeit zu einer Rolle erwachsen.
Die Rollen können hierarchisch organisiert sein und werden Benutzern vom Systemadministrator zugewiesen. Für die Zusammenhänge zwischen Benutzern, Rollen und Transaktionen gibt es verschiedene Regeln, die deren freie Kombinierbarkeit einschränken.

Im Unterschied zu Zugriffskontrollisten und dem Vertraulichkeitsmodell stellen rollenbasierte Zugriffskontrollen den Schutz der Integrität in den Vordergrund, wobei Vertraulichkeit durch eine entsprechende Implementation des Modells ebenfalls gewährleistet werden kann. Eine Weiterentwicklung des rollenbasierten Zugriffskontrollmodells beschreibt [Schier 99].

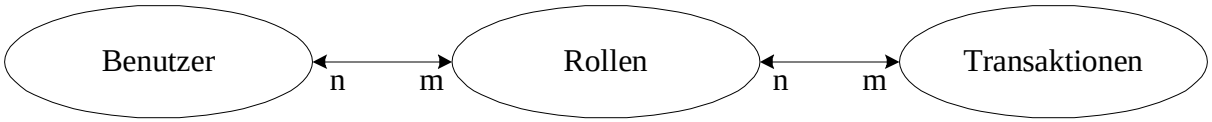


Abbildung 6: Beziehung zwischen Benutzern, Rollen und Transaktionen (nach [Ferraiolo, Cugini, Kuhn 1995])

Ein Beispiel für ein rollenbasiertes Zugriffskontrollsystem für ein Hotel findet sich in [Beis, Vorwerk 97]. Dort wurde ein chipkartengestütztes, rollenbasiertes Zugangssystem für das Hotelgewerbe implementiert. Einige Rollen sind „Administrator“, „Raumpflege“, „Verwaltung“ und „Personal“. Einer Rolle sind eine oder mehrere Transaktionen auf einem Objekt zugeordnet. Im Beispiel sind die Transaktionen Zugriffsrechte auf verschiedene Objekte. So darf z.B. ein Angehöriger der Rolle „Personal“ die Garage benutzen. Durch die hierarchische Anordnung der Rollen ist der Administrator gleichzeitig Mitglied aller anderen Rollen und darf daher deren Transaktionen ausführen (siehe Abbildung 7).

Rollenbasierte Modelle gehen im allgemeinen weiter als andere Zugriffskontrollmodelle und können mehrere Sicherheitsanforderungen abdecken. Einfache rollenbasierte Zugriffsmodelle lassen sich durch diskretionäre Zugriffskonzepte realisieren (siehe z.B. [Beis, Vorwerk 97]), komplexere Rollenmodelle können so aber nicht abgebildet werden ([Barkley 97]).

Rollenmodelle lassen sich besonders gut verwalten, da die Rechte eines Benutzers und der Benutzer selbst durch die Indirektion „Rolle“ entkoppelt sind. Ein Anwender kann z.B. benötigte Zugriffsrechte einfach erhalten, indem ein Administrator ihm eine zusätzliche Rolle zuweist. Aufgrund der Einbeziehung von Transaktionen in das Modell läßt sich die Integrität von Objekten schützen, was ein klarer Vorteil gegenüber den vorgestellten diskretionären Modellen ist.

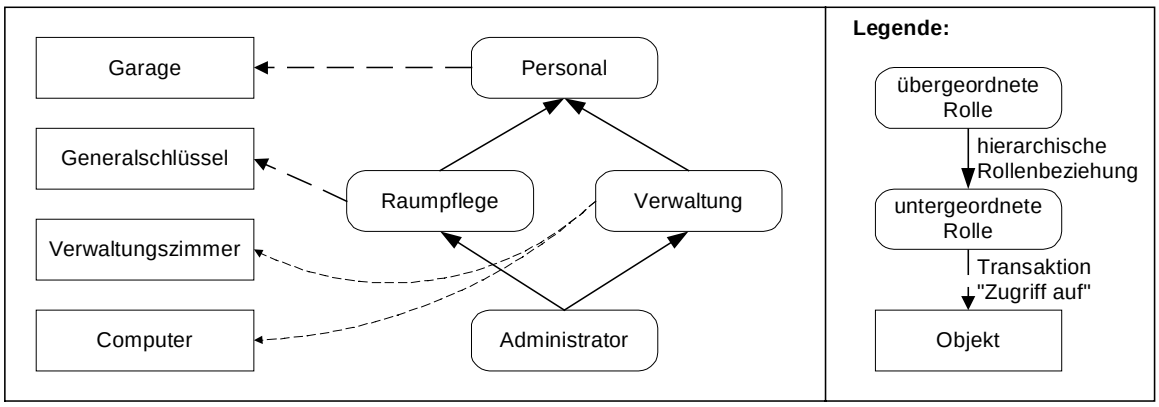


Abbildung 7: Rollenbasierte Zugriffskontrolle mit multifunktionalen Chipkarten in einem Hotel (nach [Beis, Vorwerk 97])

Fachliche und technische Zugriffskontrollen

Die vorgestellten Zugriffskontrollmodelle befassen sich allgemein mit dem Zugriff auf Objekte. In Abschnitt 2.1.2 wurde bereits angeführt, daß Sicherheit sowohl auf fachlicher Ebene als auch auf technischer Ebene betrachtet werden kann. Diese Betrachtung läßt sich auf Zugriffskontrollen ausdehnen. Als Beispiel hierfür wird eine Kundenkartei betrachtet:

> *Beispiel:*
> Innerhalb eines Anwendungssystems existiert eine Kundenkartei. Diese enthält Informationen über die Kunden eines Unternehmens. Die Kundenkartei sollte von allen Mitarbeitern mit Kundenkontakt eingesehen und gepflegt werden können. Mitarbeiter ohne Kundenkontakt benötigen keinen Zugriff.

Zugriffskontrollen auf die Kundenkartei sind fachlich motiviert. Auf softwaretechnischer Ebene werden sie durch eine fachliche Zugriffskontrolle im Anwendungssystem realisiert. Die Kundenkartei existiert aber nicht nur zur Laufzeit im Anwendungssystem, sondern wird mit einem *Persistenzmedium*[8] (z.B. einer Datei) synchronisiert (siehe Abbildung 8).

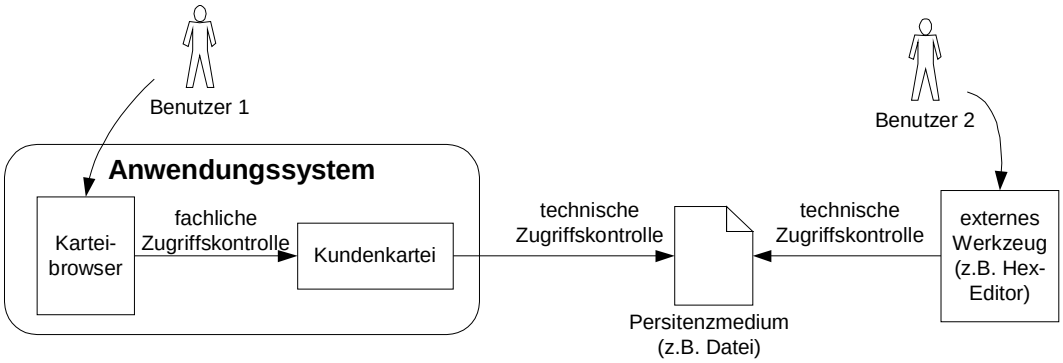


Abbildung 8: Fachliche und technische Zugriffskontrollen

Die Zugriffe auf die Datei können nicht nur innerhalb des Anwendungssystems erfolgen, sondern auch durch externe Werkzeuge (z.B. einen Hex-Editor, mit dem der Inhalt von Dateien auf Hexadezimalcode-Ebene betrachtet und manipuliert werden kann). Um die Sicherheit der Kundenkartei zu gewährleisten, müssen die *fachlichen Zugriffskontrollen* durch *technische Zugriffskontrollen* auf das Persistenzmedium ergänzt werden. Zur Unterscheidung der Begriffe wird festgehalten:

> **Definition 15: Fachliche Zugriffskontrollen**
> Fachliche Zugriffskontrollen regeln den Zugriff auf anwendungsfachliche Gegenstände innerhalb eines Anwendungssystems.

> **Definition 16: Technische Zugriffskontrollen**
> Technische Zugriffskontrollen regeln den Zugriff auf technische Objekte (z.B. eine Datei oder Datenbank).

Fachliche und technische Zugriffskontrollen können mit beliebigen Zugriffskontrollmodellen realisiert werden. Die Unterscheidung in fachliche und technische Zugriffskontrollen bezieht sich lediglich darauf, auf welcher Ebene der Zugriffsschutz ansetzt. Beide Arten von Zugriffskontrollen müssen sinnvoll miteinander kombiniert werden, um Zugriffsschutz für den zu schützenden Gegenstand zu gewährleisten.

---

[8] Der Begriff *Persistenz* wird in der Softwaretechnik wie folgt verstanden: „Speicherung von Objekten und Datenstrukturen, die das Konvertieren komplexer Datenstrukturen in ein Format beinhaltet, das für die Dateispeicherung geeignet ist. Persistente Daten werden von einem Computer zwischen Sitzungen beibehalten." (Quelle: [MSDN 99b]). Ein Persistenzmedium ist z.B. eine Datenbank.

2.3. Zugriffskontrollen für WAM-Systeme

Wie in den letzten Abschnitten deutlich wurde, gibt es eine Reihe verschiedener Zugriffskontrollmodelle. In diesem Abschnitt soll untersucht werden, welches Zugriffskontrollmodell sich für WAM-Systeme eignet. Normalerweise läßt sich diese Diskussion nur in Bezug auf ein konkretes Anwendungssystem führen, da Sicherheitsanforderungen immer für einen konkreten Anwendungsbereich ermittelt werden müssen (vergleiche Abschnitt 2.1.2). Auf einer allgemeinen Ebene läßt sich nur diskutieren, weil alle WAM-Systeme nach einer gemeinsamen Idee und einem ähnlichen Entwicklungsprozeß entworfen werden.

2.3.1. Überblick über den WAM-Ansatz

Eine „zentrale Idee“ des WAM-Ansatzes ist, „die Gegenstände und Konzepte des Anwendungsbereichs als Grundlage des softwaretechnischen Modells zu nehmen“ ([Züllighoven et al. 98], Seite 4f). Anwender finden so „Gegenstände ihrer Arbeit und die Begriffe ihrer Fachsprache im Anwendungssystem repräsentiert“ und die „Entwickler können Softwarekomponenten und Anwendungskonzepte bei fachlichen und softwaretechnischen Änderungen zueinander in Beziehung setzen und somit die wechselseitigen Abhängigkeiten erkennen“ ([Züllighoven et al. 98], Seite 5). Handelt es sich bei dem Anwendungsbereich zum Beispiel um eine Bank, so wird es in der Softwarearchitektur sicherlich ein Modell eines Kontos geben. Geprägt wird der softwaretechnische Entwurf durch ein sogenanntes *Leitbild*:

> **Definition 17: Leitbild**
> Ein Leitbild ist eine benennbare, grundsätzliche Sichtweise, anhand derer wir einen Ausschnitt von Realität wahrnehmen, verstehen und gestalten. Ein Leitbild repräsentiert immer auch eine Wertvorstellung. (aus [Züllighoven et al. 98], Seite 73f)

Durch die Verwendung der Begriffe „wahrnehmen“ und „Wertvorstellung“ zeigt bereits diese Definition, daß ein softwaretechnischer Entwurf im WAM-Ansatz abhängig von den beteiligten Personen ist. Neben Auftraggebern und Anwendungsentwicklern handelt es sich dabei auch um die Anwender des zu entwerfenden Anwendungssystems. Diese beteiligten Personen nehmen eine Sichtweise ein, die entscheidenden Einfluß auf die Gestalt des zukünftigen Anwendungssystems hat.
Das im WAM-Ansatz am häufigsten verwendete Leitbild ist das Leitbild vom „Arbeitsplatz für eigenverantwortliche Expertentätigkeit“ ([Züllighoven et al. 98], S. 80). Hierbei handelt es sich beispielsweise um den Arbeitsplatz eines persönlichen Kundenberaters in einer Bank. Dieses Leitbild betont, daß es sich bei den Anwendern um hochqualifizierte Experten handelt, die ihren Arbeitsablauf selbst bestimmen. Da der Kundenberater über ein hohes Maß an Erfahrung und Entscheidungskompetenz verfügt, darf ein unterstützendes System seinen Arbeitsfluß nicht bestimmen. Auch wenn für ihn bestimmte Regeln für die Geschäftsabläufe gelten, ist er frei in ihrer Auslegung.

Für ein Anwendungssystem bedeutet dies, daß statt einer Ablaufsteuerung eine Unterstützung integriert werden muß (vergl. [Züllighoven et al. 98], S. 81 und [Gryczan 96]). Diese Unterstützung schafft eine „geeignete Arbeitsumgebung“, die die in einer konkreten Situation benötigten Arbeitsgegenstände zur Verfügung stellt ([Züllighoven et al. 98], S. 81).

Konkretisiert wird das Leitbild durch Entwurfsmetaphern. Diese geben Hinweise, wie sich einzelne anwendungsfachliche Konzepte sinnvoll voneinander abgrenzen lassen und wie sie im Anwendungssystem abgebildet werden können:

> **Definition 18: Entwurfsmetapher**
> Eine Entwurfsmetapher ist eine bildhafte, gegenständliche Vorstellung, die ein Leitbild fachlich und konstruktiv „ausgestaltet“, d.h. konkretisiert. Eine Entwurfsmetapher strukturiert die Wahrnehmung und trägt zur Begriffsbildung bei. Sie leitet die Vorstellung und Kommunikation über das, was fachlich analysiert, modelliert und technisch realisiert werden soll. (siehe [Züllighoven et al. 98], S. 79)

Die in diesem Zusammenhang wichtigsten Entwurfsmetaphern sind Material, Werkzeug, Behälter und Arbeitsumgebung (mehr hierzu in Kapitel 3).
Die Entwurfsmetaphern passen gut in die Arbeitswelt des Anwenders und bilden softwaretechnische Abstraktionen über dort vorgefundene Gegenstände und Konzepte (wie z.B. die *Materialien* Konto und Überweisung). Ein WAM-System beinhaltet daher im wesentlichen die softwaretechnischen Modelle von Konzepten und Gegenständen des Anwendungsbereiches. Die Entwurfsmetaphern helfen, diese Modelle zu strukturieren.

2.3.2. Einordnung von WAM-Anwendungssystemen

Zwei wichtige Bestandteile des WAM-Ansatzes sind die Beteiligung der Anwender am Entwurfsprozeß und der Gedanke, Anwender durch das Anwendungssystem zu unterstützen. Die entscheidende Bedeutung dieser beiden Punkte für WAM-Systeme wird deutlich, wenn man die Klassifikation von Anwendungssystemen nach M. Lehman betrachtet (siehe [Lehman 80]). Lehman unterscheidet zwischen S (*specification*), P (*real world problem*) und E (*embedded*)- Programmen (vgl. Abbildung 9).

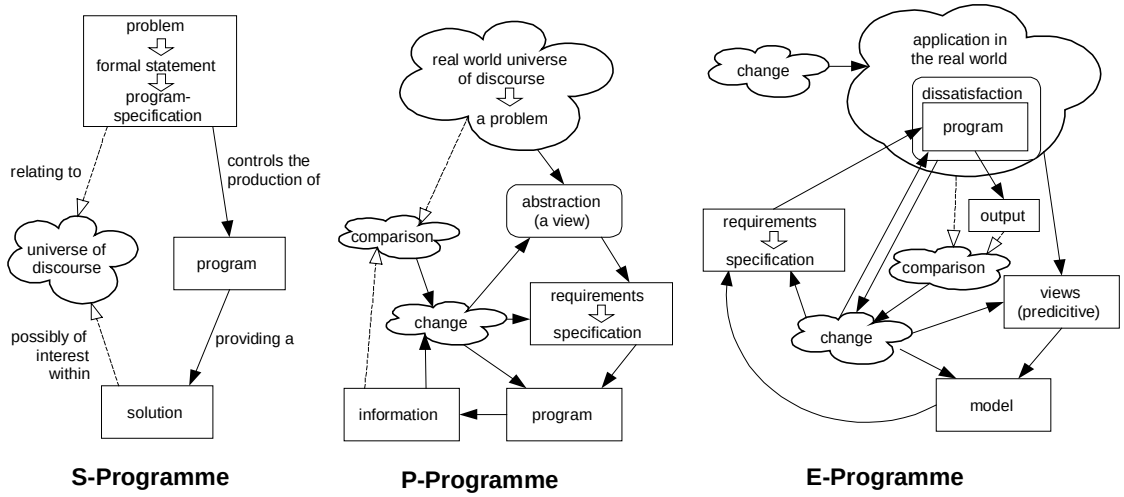


Abbildung 9: Klassifikation von Programmen nach Lehman (aus [Lehman 80])

S-Programme sind Implementationen vollständig spezifizierbarer Probleme. Die Spezifikation ist personenunabhängig und das Programm kann durch mathematische Methoden verifiziert werden. Ein Beispiel ist die Berechnung von Zinseszinsen.

P-Programme basieren ebenfalls auf einem vollständig spezifizierbaren Problem. Allerdings müssen aufgrund der Menge an Eingabedaten sowie der Komplexität der Berechnungen Abstraktionen vorgenommen werden, so daß die eigentliche Spezifikation den Standpunkt ihres Entwicklers und damit subjektive Elemente beinhaltet. Beispiele für P-Programme sind Schachprogramme. Entspricht das Ergebnis nicht den Erwartungen (Beispiel: das Schachprogramm verliert zu häufig), werden die Anforderungen und entsprechend das Programm geändert (z.B. durch Veränderung der Bewertung von Figur-Stellungen anhand von Erfahrungswerten).

E-Programme sind Anwendungssysteme, die Teile eines Anwendungsbereichs modellieren und zur Ausführungszeit selbst Bestandteil des Anwendungsbereichs werden: „the program has become a part of the world it models, it is embedded in it" ([Lehman 80]). E-Programme werden entwickelt, um Unterstützung in konkreten Arbeitssituationen, zu bieten ([Züllighoven et al. 98], S. 552). Da das Programm zum Bestandteil des modellierten Ausschnitts eines Anwendungsbereichs wird, müssen auch die Auswirkungen seiner Einführung in den Anwendungsbereich berücksichtigt werden. Beispiele für solche Systeme sind Krankenhausinformationssysteme.

Entwicklung von E-Programmen

Ein E-Programm ist ein Modell eines Anwendungsbereichs. In einem solchen Modell werden die für wesentlich erachteten Teile des Anwendungsbereichs berücksichtigt (vgl. [Schier 99], Seite 127). Welche Teile des Anwendungsbereichs für wesentlich erachtet werden, ist eine subjektive Entscheidung der am Entwicklungsprozeß Beteiligten. Um zu einem brauchbaren Programm zu gelangen, ist es notwendig, „daß die Entwickler den nicht-formalisierbaren Kontext und die Arbeitszusammenhänge eines Anwendungssystems verstanden haben“ ([Züllighoven et al. 98], S. 552). Dieser Kontext und die Arbeitszusammenhänge können den Anwendungsentwicklern nur durch die Anwender selbst vermittelt werden. Damit unterliegt die Problembeschreibung eines E-Programms subjektiven Einflüssen.

Besonders für E-Programme gilt daher, daß Entwurf und Design eines Anwendungssystems entscheidend durch die beteiligten Personen geprägt werden (vgl. [Züllighoven et al. 98], S. 549ff). Jeder Anwender und jeder Anwendungsentwickler hat eine andere Vorstellung davon, was ein Problem ist und wie die Unterstützung einer Arbeitssituation aussehen kann. Entsprechend kann das Anwendungssystem als Ergebnis des Entwurfes nicht „falsch“ oder „richtig“ sein, sondern ist vielmehr „geeignet“ oder „weniger geeignet“ zur Unterstützung eines Arbeitsplatzes (siehe auch [Floyd 94]).

Bei WAM-Anwendungssystemen handelt es sich um interaktive Anwendungssysteme mit unterstützendem Charakter für qualifizierte Experten. Um diese Unterstützung leisten zu können, müssen die Systeme in den Arbeitszusammenhang im Anwendungsbereich eingebettet sein. Die nach diesem Leitbild entworfenen Systeme lassen sich daher als E-Programme einordnen (siehe auch [Züllighoven et al. 98], Seite 552f).

Die genannten subjektiven Einflüsse müssen im WAM-Entwicklungsprozeß berücksichtigt werden. Dies geschieht durch eine enge Zusammenarbeit von Anwendern und Anwendungsentwicklern während des gesamten Entwicklungsprozesses. Vor diesem Hintergrund gewinnt die Kommunikation zwischen beiden Gruppen eine entscheidende Bedeutung. Das Ziel ist die permanente Rückkopplung von Arbeitsergebnissen.

Als Arbeitsergebnisse erhalten die Anwender Prototypen und für sie verständliche Entwurfsdokumente, die sich der Fachsprache der Anwender bedienen. Auf diese Weise lernen Anwender (und auch Auftraggeber) das Anwendungssystem frühzeitig und schrittweise kennen und können Einfluß auf seine weitere Entwicklung nehmen. Die Anwendungsentwickler erfahren, welche Anforderungen an das System sie noch berücksichtigen müssen und an welchen Stellen Probleme im Umgang mit dem System auftreten.

Korrektheit von E-Programmen

Lehman weist außerdem darauf hin, daß es erstrebenswert sei, ein größeres P- oder E-Programm in kleinere S-Programme zu zerlegen, so daß auch größere Software spezifizierbar werde. Allerdings stellt Lehman selbst die Relevanz einer solchen formalen Spezifikation für E-Programme in Frage:

> „Correctness and proof of correctness of the program as a whole are, in general, irrelevant in that a program may be formally correct but useless, or incorrect in that it does not satisfy some stated specification, yet quite usable, even satisfactory.“ (siehe [Lehman 80])

Ebenfalls zweifelhaft ist, daß eine Zerlegung eines E-Programmes in formal spezifizierbare S-Module überhaupt gelingen kann. Die genannten subjektiven Einflüsse auf den Entwicklungsprozeß sind einer formalen Spezifikation nicht zugänglich und E-Programme lassen sich daher nicht vollständig in S-Module zerlegen. Demzufolge können Sicherheitseigenschaften nur für die S-Module eines E-Programmes, nicht jedoch für das ganze E-Programm, formal verifiziert werden. Ähnlich argumentiert auch Schier: „(...) bei der Umsetzung der Wirklichkeit in ein Modell [kann] kein vollständiger Beweis für die Sicherheit des Systems erbracht werden“ ([Schier 99], S. 127).

Es sei darauf hingewiesen, daß eine formale Spezifikation von S-Modulen innerhalb eines E-Programms durchaus Sinn machen kann. Ein Beispiel hierfür ist ein Anwendungssystem, welches Chipkarten zur Zugangskontrolle verwendet. Das Authentifikations-Modul mit der Chipkartenanbindung kann vollständig spezifiziert und formal verifiziert werden.

2.3.3. Anforderungen an Zugriffskontrollen in WAM-Systemen

Es wird davon ausgegangen, daß alle vorgestellten Zugriffskontrollmodelle (unabhängig von einer konkreten Implementation) prinzipiell dazu geeignet sind, Anwendungssysteme durch Zugriffskontrollen abzusichern. Jedes dieser Modelle hat seine Stärken und Schwächen und ist für verschiedene Einsatzkontexte unterschiedlich gut geeignet. Diese Arbeit soll eine Antwort auf die Frage liefern, wie ein geeignetes Zugriffskontrollmodell für WAM-Systeme aussehen soll. Daher soll heraus gearbeitet werden, welche Aspekte eines Zugriffskontrollmodells für WAM-Systeme besonders relevant sind.

Hierzu wird zuerst eine Studie von A. Adams und M. Sasse betrachtet ([Adams, Sasse 99]). In dieser Studie wurde untersucht, wie Anwender in kommerziell orientierten Organisationen mit Paßwörtern umgehen. Dabei wurde festgestellt, daß Anwender durch ihren Umgang mit den Paßwörtern die Sicherheitsmechanismen der Organisation mehr oder minder bewußt kompromittieren. Eine Ursache hierfür sei demnach, daß Sicherheitsmaßnahmen oft inkompatibel zu gewohnten Arbeitsabläufen waren. Sicherheitsabteilungen wüßten wenig über die Arbeit der Anwender und Anwender wiederum verstünden Sinn und Funktion vieler Sicherheitsmechanismen nicht. Als Folge umgingen Anwender diese Maßnahmen. Die Sicherheitsabteilungen wiederum betrachteten Anwender als „inhärent unsicher“ und schufen neue Sicherheitsmechanismen. Als Ergebnis fordern Adams und Sasse in der zitierten Arbeit, daß Anwender in die Gestaltung der Sicherheitsmechanismen einbezogen werden sollen.

Die in der Studie betrachteten Anwendungssysteme sind offenbar in einen Anwendungsbereich eingebettet und damit handelt es sich um E-Programme. Die Sicherheitsabteilungen haben anscheinend nicht berücksichtigt, daß durch E-Programme Rückwirkungen auf den Anwendungsbereich auftreten. Anwender verändern ihren Arbeitsablauf, um mit dem Anwendungssystem besser umgehen zu können (siehe [Lehman 80]).

> **Ergebnis 1**
> Die Integration von Sicherheitsmaßnahmen in E-Programme hat Rückwirkungen auf den Anwendungsbereich. Diese Rückwirkungen müssen bei der Einrichtung und Durchsetzung von Sicherheitsmaßnahmen berücksichtigt werden.

Offensichtlich läßt sich aus der Integration von Anwendern in den Entwicklungsprozeß auch in Sicherheitsfragen ein Nutzen ziehen. Hierzu ist der Dialog mit den Anwendern nötig (vgl. Abschnitt 2.3.2), und damit diese Kommunikation möglich wird, müssen die Anwender das grundlegende Sicherheitsmodell eines Anwendungssystems verstehen können.

> **Ergebnis 2**
> Bei der Entwicklung eines Sicherheitsmodells sollen Anwender in den Entwurfsprozeß einbezogen werden. Hierzu muß das Sicherheitsmodell für Anwender verständlich sein.

Weiterhin ist wichtig, daß sich das Sicherheitsmodell im Arbeitsalltag bewährt und einfach zu handhaben ist (vgl. *einfache Handhabung*, Definition 6), damit der Arbeitsablauf nur minimal beeinträchtigt wird. Da die unterstützende Sichtweise für WAM-Systeme eine große Rolle spielt, darf das Zugriffskontrollmodell den Arbeitsablauf nicht vorgeben. Anwender sollen ebenfalls die Möglichkeit haben, ihre Arbeitsumgebung nach Belieben einzurichten ([Züllighoven et al. 98], S. 178). Dies gilt sowohl für die individuelle als auch für gemeinschaftlich genutzte Arbeitsumgebungen. Gerade für Arbeitsgruppen ist es wichtig, private und gemeinsam genutzte Arbeitsräume selbständig mit Zugriffskontrollen abzusichern.

Ergebnis 3
WAM-Systeme sind in einen Anwendungsbereich eingebettet und sollen die Arbeit der Anwender unterstützen. Sicherheitsmechanismen sollten diesen unterstützenden Charakter nach Möglichkeit nicht beeinträchtigen und Anwendern ermöglichen, Zugriffskontrollen einem privaten oder für sie überschaubaren Kontext selbst zu definieren.

Bei WAM-Systemen handelt es sich um kundenspezifische Entwicklungen, die an die jeweilige Organisation und dort an verschiedene Arbeitsplätze angepaßt werden müssen. Das Zugriffskontrollmodell muß hinreichend flexibel sein, um verschiedene Anwendungsbereiche unterstützen zu können und sich dabei dynamisch auf veränderliche Arbeitsabläufe innerhalb der Organisation abbilden lassen.

Ergebnis 4
WAM-Systeme können in sehr unterschiedlichen Anwendungsbereichen eingesetzt werden. Ein Zugriffskontrollmodell darf diese Flexibilität nicht beschränken.

Innerhalb eines WAM-Systems lassen sich die modellierten Gegenstände und Konzepte des Anwendungsbereichs den genannten Entwurfsmetaphern zuordnen (siehe Abschnitt 2.3.1). Dadurch lassen sich die Elemente des softwaretechnischen Modells und ihre Beziehungen zueinander auf einer Ebene oberhalb des Objekt-Metamodells[9] diskutieren. Dies gilt insbesondere für eine Absicherung dieser Elemente durch fachliche Zugriffskontrollen.

Ergebnis 5
In WAM-Systeme werden anwendungsfachliche Gegenstände und Konzepte mit Hilfe von Entwurfsmetaphern modelliert. Ein Zugriffskontrollmodell für WAM-Systeme sollte zu diesem Modell passen und es unterstützen.

Aus den genannten Ergebnissen lassen sich die folgenden speziellen Anforderungen für eine Zugriffskontrolle in WAM-Systemen formulieren. Das Zugriffskontrollmodell soll

- verständlich für Anwender sein,
- den unterstützenden Charakter von WAM-Systemen betonen,
- die Flexibilität des WAM-Ansatzes nicht einschränken und
- durch Entwurfsmetaphern modellierte, softwaretechnische Gegenstände absichern.

Die genannten Anforderungen beziehen sich auf fachliche Zugriffskontrollen innerhalb eines Anwendungssystems. Welche Eigenschaften ein hierzu passendes Zugriffskontrollmodell für technische Zugriffskontrollen haben sollte, wird durch sie nicht ausgedrückt.

2.3.4. Überprüfung der Anforderungen

Nachdem die Anforderungen für ein fachliches Zugriffskontrollmodell für WAM-Systeme herausgearbeitet wurden, sollen nun die in Abschnitt 2.2.2 vorgestellten Zugriffskontrollmodelle daraufhin überprüft werden, inwieweit sie diese Anforderungen erfüllen.

Verständlichkeit für Anwender
Diese Forderung wird von Zugriffskontrollisten nur eingeschränkt erfüllt. Sie stellen für die Anwender ein aus dem Arbeitsalltag nicht geläufiges Konzept dar, welches erlernt werden muß. Zugriffskontrollisten werden aus diesem Grund nur von technisch versierten Anwendern verstanden und verwendet.

Schlüssel und Schlösser sind ein aus dem täglichen Leben vertrautes Konzept, mit dem Anwender ohne weiteres umgehen können. Schlüssel und Schlösser im SKL-Modell wurden

[9] „Ein Objekt-Metamodell beschreibt die Elemente und deren Verknüpfungen, die für die objektorientierte Modellierung zur Verfügung stehen“, ([Züllighoven et al. 98], Seite 19 ff).

jedoch nicht als anwendungsfachliche Gegenstände modelliert, sondern auf eine pure mathematische Funktionalität reduziert. Damit ist dieses Modell als nicht ausreichend verständlich einzuschätzen. Es kann daher nur technisch versierten Anwendern nahegebracht werden.

Mandatorische und rollenbasierte Konzepte sind mit ihren Klassifikationen bzw. Rollen und Aufgaben an den Bedürfnissen einer Organisation orientiert. Besonders Anwendern aus einem militärischen Umfeld sollten die Klassifikationen des Vertraulichkeitsmodells bekannt sein. Anwender aus einem anderen Anwendungsbereich – und um solche geht es bei WAM-Systemen im wesentlichen – haben diesen Vorteil nicht. Ihnen hilft zum Verständnis des Modells lediglich die allgemein verständliche Bezeichnung der Klassifikationen.

Wie verständlich Rollenmodelle für Anwender sind, hängt von ihrer konkreten Umsetzung ab. Rollen bilden Abstraktionen über die tägliche Arbeit von Anwendern. Anwender müssen diese Abstraktionen verstehen, um sinnvoll über Rollenmodelle diskutieren zu können. Eine geeignete Möglichkeit diese Abstraktionen zu bilden, ist die Verwendung von *use cases* und *actors* (siehe [Jacobson 92], [Booch et al. 99]). Zusammenfassend läßt sich der grundlegende Gedanke dieses Konzeptes durch folgendes Zitat beschreiben:

> „A use case describes a set of sequences, in which each sequence represents the interaction of things outside the system (its actors) with the system itself. (...) For example, one central use case of a bank is to process loans. (...) An actor represents a coherent set of roles that users of use cases play when interacting with these use cases.“ ([Booch et al. 99], Seite 220f)

Use cases werden in Entwurfsdokumenten beschrieben, die mit Anwendern diskutiert werden. Die actors können individuelle Personen oder auch Rollen sein. Im Entwurfsprozeß können sich aus den Aufgaben einzelner Personen als Abstraktion Rollen herausbilden.

Ergebnis 6
Zugriffskontrollisten und das SKL-Modell sind zu technisch orientiert, um Anwendern nahegebracht zu werden. Das Vertraulichkeitsmodell ist für Anwender hinreichend verständlich. Bei rollenbasierten Ansätzen hängt die Verständlichkeit sehr vom konkret verwendeten Entwurfsprozeß ab. Daher kann hier keine allgemeine Aussage bezüglich ihrer Verständlichkeit formuliert werden.

Flexibilität
Aufgrund ihrer Einfachheit sind diskretionäre Zugriffskontrollen in jedem Einsatzkontext verwendbar, allerdings gibt es Regelungen, die ihre Verwendung in besonders sicherheitskritischen Umgebungen ausschließen (z.B. [DoD 1985], [ITSEC 1991][10]). Bisherige WAM-Anwendungssysteme unterliegen diesen Anforderungen nicht. Aufgrund der genannten Schwierigkeiten im Zusammenhang mit der Administrierbarkeit von Zugriffskontrollisten (siehe Abschnitt 2.2.2) sollten diese in großen Anwendungssystemen nicht verwendet werden.

Das Vetraulichkeitsmodell ist eher statisch als flexibel. Einmal klassifizierte Objekte können nicht mehr in ihrer Klassifikation heruntergesetzt werden, was bei veränderten Geschäftsabläufen und im Arbeitsalltag ein Problem darstellt. Darüber hinaus sind Art und Anzahl der möglichen Transaktionen nicht dynamisch erweiterbar.

Bei rollenbasierter Zugriffskontrolle ist diese Flexibilität hingegen gegeben. Ist das Modell für einen Anwendungsbereich konkretisiert worden, lassen sich zum Beispiel einfach neue Transaktionen hinzufügen oder Transaktionen neuen Rollen zuweisen und damit der veränderte Geschäftsablauf nachbilden.

[10] Eine Diskussion der in den angegebenen Werken vorgestellten Sicherheitsrichtlinien ist für diese Arbeit nicht weiter relevant und unterbleibt daher.

> **Ergebnis 7**
> Sowohl diskretionäre als auch rollenbasierte Ansätze erweisen sich als hinreichend flexibel für den Einsatz in verschiedenen Umgebungen. Zugriffskontrolllisten sollten in größeren Systemen nicht eingesetzt werden. Das Vetraulichkeitsmodell erweist sich als zu unflexibel für nicht-militärische Anwendungsbereiche.

Betonung der unterstützenden, eigenverantwortlichen Sichtweise
Diskretionäre Modelle beschränken die unterstützende, eigenverantwortliche Sichtweise eines Anwendungssystems nicht. Zudem erlauben sie die selbständige Definition von Zugriffskontrollen durch Anwender in einem privaten oder für sie überschaubaren Kontext. In diesem Zusammenhang wäre ein einfaches, leicht verständliches Zugriffskontrollmodell von Vorteil. Auch hier zeigt sich, daß die wenig intuitive Umsetzung der Schlüssel-Schloß Metapher im SKL-Modell nicht die Stärken der Metapher berücksichtigt.

Bei WAM-Systemen macht sich der unterstützende Charakter dadurch bemerkbar, daß der Anwender frei in der Auswahl seiner (Trans)Aktionen ist. Prämisse ist, daß der Anwender weiß, was er tut. Speziell muß es möglich sein, die Tätigkeiten innerhalb einer Aufgabe in einer beliebigen Reihenfolge auszuführen. Bezogen auf die Entwurfsmetaphern Werkzeug und Material heißt dies, daß ein Anwender ein verfügbares Material immer bearbeiten kann, wenn er es prinzipiell bearbeiten darf. Dies widerspricht dem Gedanken wohlgeformter Transaktionen, die einzelne Ausführungsschritte in einer bestimmten Reihenfolge anordnen. Mandatorische Modelle können die freie Kombinierbarkeit von Tätigkeiten stärker einschränken als diskretionäre Ansätze.

Aus der Definition mandatorischer und rollenbasierter Modelle folgt, daß Anwender Zugriffskontrollen nicht selbst konfigurieren können (siehe Abschnitt 2.2.2).

Vor dem Hintergrund einer unterstützenden Sichtweise kann die Kontrolle des Informationsflusses (vgl. Abschnitt 2.2.2) durch mandatorische Modelle auch Nachteile bringen. Wenn ein Anwender wünscht, einen anderen Anwender über einen Sachverhalt in Kenntnis zu setzen, sollte er nicht vom System daran gehindert werden. Allerdings ist dann auch kein Schutz vor unbeabsichtigten Informationsflüssen mehr vorhanden.

> **Ergebnis 8**
> Diskretionäre Modelle schränken den unterstützenden Charakter eines Anwendungssystems nicht maßgeblich ein und erlauben selbstdefinierte Zugriffskontrollen. Mandatorische und rollenbasierte Ansätze sind geeignet, den Arbeitsablauf vorzugeben und damit die Unterstützung durch ein Anwendungssystem zu beschränken. Sie erlauben keine selbstdefinierten Zugriffskontrollen.

Absicherung von softwaretechnischen Gegenständen
Vom reinen Modell her paßt lediglich die Schlüssel-Schloß-Metapher gut zu anderen Entwurfsmetaphern. So ließen sich beispielsweise Anwender mit verschiedenen Schlüsseln und Behälter mit Schlössern ausstatten. Da es sich bei konkreten Behältern dann um anwendungsfachliche Gegenstände handelt, ist auch klar, wo Schlösser angebracht werden müßten und wie der entsprechende Umgang mit den Behältern wäre.

Ein Beispiel hierfür wäre ein Briefkasten: ein Anwender kann Post in den Briefkasten eines anderen Anwenders werfen, aber nur der Besitzer eines Schlüssels kann den Briefkasten öffnen und die Post wieder herausnehmen. Die Post selbst muß dann nicht mehr gegen unberechtigte Zugriffe gesichert werden.

Aufgrund seiner technischen Orientierung ist das SKL-Modell zum Schutz softwaretechnischer Gegenstände ungeeignet.

Ergebnis 9
Die gezielte Absicherung von Gegenständen im softwaretechnischen Modell kann von den vorgestellten Zugriffskontrollmodellen nicht geleistet werden. Eine mehr anwendungsfachlich gefaßte Definition und Umsetzung der Schlüssel-Schloß-Metapher scheint zur Absicherung von WAM-Konzepten geeignet zu sein.

2.3.5. Beurteilung

Tabelle 1 faßt die Diskussion des vorangegangenen Abschnittes zusammen. Dabei werden als Bewertungen *ungeeignet* (–), *geeignet* (o) und *gut geeignet* (+) verwendet:

	Zugriffskontroll-Listen	**Single-Key-Lock Ansatz**	**BLP Vertraulichkeits-Modell**	**RBAC**
Verständlichkeit	–	–	o	keine Aussage möglich
Flexibilität	o	+	–	+
unterstützende Sichtweise	+	+	o	o
Schutz softwaretechn. Gegenstände	–	–	–	–

Tabelle 1: Bewertung von Zugriffskontrollmodellen für WAM-Systeme

Unter den vorgestellten Zugriffskontrollmodellen gibt es offensichtlich keines, das die speziellen Anforderungen von WAM-Systemen hinreichend erfüllt:

Ergebnis 10
Die untersuchten Zugriffskontrollmodelle können die Anforderungen für fachliche Zugriffskontrollen in WAM-Systemen nicht befriedigend erfüllen.

Es sei darauf hingewiesen, daß diese Bewertungen vor dem Hintergrund einer besonderen Klasse von Anwendungssystemen gelten. Alle Modelle haben sich in anderen Einsatzkontexten durchaus bewährt. Insbesondere die Anforderung nach der „Absicherung von WAM-Konzepten“ kann kaum durch Modelle erfüllt werden, die nicht aus dem WAM-Ansatz heraus entwickelt wurden. Da es in dieser Arbeit um die Absicherung von WAM-Systemen geht, ist diese Anforderung sehr relevant.

Die zuletzt in einem Beispiel verdeutlichte Verwendung der Schlüssel-Schloß-Metapher bietet eine interessante Alternative zu den bisher vorgestellten Zugriffskontrollmodellen. Sie ist Anwendern und Anwendungsentwicklern aus dem täglichen Leben vertraut und läßt sich vor allem gut im Zusammenhang mit anderen anwendungsfachlichen Gegenständen (wie z.B. Behältern) verwenden. Notwendig ist eine Interpretation der Schlüssel-Schloß-Metapher, in der der gegenständliche Charakter von Schlüsseln und Schlössern deutlich wird. Dann läßt sich auch diskutieren, wie Schlüssel und Schlösser konkret zu den anderen Entwurfsmetaphern passen und wie flexibel einsetzbar sie bei der Verwendung anderer Leitbilder sind.

Ansatz
Im weiteren Verlauf dieser Arbeit soll untersucht werden, wie sich eine neu definierte Schlüssel-Schloß-Metapher zur Zugriffskontrolle in WAM-Systemen eignet.

Dabei bleibt die Schlüssel-Schloß-Metapher ein diskretionäres Zugriffskontrollmodell, und unterliegt prinzipiell den genannten Beschränkungen:

- Eine Kontrolle des Informationsflusses (siehe Abschnitt 2.2.2) ist mit dem Schlüssel-Schloß-Prinzip nicht möglich aber auch nicht erwünscht, da bei WAM-Systemen der unterstützende Charakter im Vordergrund steht (vgl. Abschnitt 2.3.3).
- In Bezug auf die in Abschnitt 2.3.3 genannten Regelungen bezüglich des Einsatzes in besonders sicherheitssensiblen Anwendungsbereichen wurde bereits erwähnt, daß WAM-Systeme in solchen Umgebungen bisher nicht zum Einsatz kamen.

Die genannten Beschränkungen für diskretionäre Zugriffskontrollen sind damit hinnehmbar. In den folgenden Kapiteln wird die Schlüssel-Schloß-Metapher als Entwurfsmetapher definiert und im Rahmen des JWAM-Rahmenwerkes gezeigt, wie die Unterstützung der Anwendungsentwickler durch die Metapher konstruktiv aussehen kann.

2.4. Zusammenfassung

In diesem Kapitel wurden grundsätzliche Sicherheitsanforderungen vorgestellt und geklärt, wie Zugriffskontrollen vor diesem Hintergrund einzuordnen sind. Zunächst wurde heraus gearbeitet, daß fachliche von technischen Zugriffskontrollen unterschieden werden können. Innerhalb eines WAM-Systems wird ein Zugriffskontrollmodell für fachliche Zugriffskontrollen benötigt. Dieses soll im Rahmen dieser Arbeit formuliert werden. Wie ein hierzu passendes technisches Zugriffskontrollmodell aussehen sollte, kann im Rahmen dieser Arbeit nicht behandelt werden. Als offene Frage verbleibt:

> **Offene Frage 1**
> Welche Anforderungen gelten für technische Zugriffskontrollen für WAM-Systeme und welches Zugriffskontrollmodell läßt sich hierfür einsetzen?

Zugriffskontrollmodelle lassen sich in diskretionäre, mandatorische und rollenbasierte Ansätze einteilen. Für diese Ansatzarten wurden exemplarisch einige Modelle vorgestellt. Dabei handelte es sich um Zugriffskontrollisten, das Single-Key-Lock Modell, das Vetraulichkeitsmodell nach Bell und LaPadula sowie das rollenbasierte Zugriffskontrollmodell nach Ferraiolo und Kuhn. Die verschiedenen Modelle wurden vor dem Hintergrund der Einflüsse des WAM-Ansatzes auf Konstruktion und Entwicklung von Anwendungssystemen daraufhin untersucht, wie gut sie sich für fachliche Zugriffskontrollen in WAM-Systemen eignen.

Von den vorgestellten Modellen erwies sich keines als hinreichend geeignet, den speziellen Anforderungen von WAM-Systemen an fachliche Zugriffskontrollen gerecht zu werden. Daher wurde festgehalten, daß als Ansatz eine neue Definition der Schlüssel-Schloß-Metapher erfolgen soll. Dieses soll die Nachteile des SKL-Modells (mangelnde Verständlichkeit für Anwender, keine Unterstützung von WAM-Konzepten) beseitigen.

Im weiteren Verlauf dieser Arbeit werden Schlüssel und Schlösser als Metaphern für den WAM-Rahmen eingeführt und untersucht, wie Anwendungsentwickler die Schlüssel-Schloß-Metapher in der Konstruktion einsetzen können.

Kapitel 3 - Die Schlüssel-Schloß-Metapher

Im vorangegangenen Kapitel wurde festgehalten, daß die vorgestellten Zugriffskontrollmodelle den speziellen Anforderungen von WAM-Systemen nicht gerecht werden. Als alternativer Ansatz wurde vorgeschlagen zu untersuchen, wie Zugriffskontrollen in WAM-Systemen mit der Schlüssel-Schloß-Metapher durchgeführt werden können.

Zu Beginn dieses Kapitels wird ein fachliches Modell eines Benutzers im Anwendungssystem motiviert und vorgestellt. Es folgt eine Zusammenfassung des Standes der Diskussion über Zugriffskontrollen in WAM-Anwendungssystemen vor Beginn dieser Arbeit. Anschließend wird die Schlüssel-Schloß-Metapher für den WAM-Kontext definiert und ihr Bezug zu den WAM-Entwurfsmetaphern analysiert. Diese Betrachtungen gehen zunächst vom Leitbild „Arbeitsplatz für eigenverantwortliche Expertentätigkeit“ aus und werden anschließend auf andere Arbeitsplatztypen sowie kooperative Arbeitssituationen ausgedehnt.

3.1. Ein fachliches Modell eines Benutzers

Im vorangegangenen Kapitel wurde deutlich, daß die Identität des Benutzers bekannt sein muß, wenn Zugriffskontrollen durchgeführt werden sollen. Damit wird der Benutzer ein Teil des Modells des Anwendungssystems. Im Rahmen des WAM-Ansatzes tritt der Benutzer auch bei anderen Konzepten als Teil des anwendungsfachlichen Modells in Erscheinung:

⇒ Die *Registratur* ist ein fachlich motiviertes Persistenzmedium, in dem Benutzer Materialien ablegen bzw. herausnehmen können (siehe Abschnitt 3.7.2). Materialien, die in der Registratur verwahrt werden, verfügen über Informationen über ihren Erzeuger, ihren letzten Bearbeiter sowie ihren Besitzer. Nimmt ein Benutzer ein Material aus der Registratur, wird seine Identität auf einer Fehlkarte vermerkt.

⇒ Die *Arbeitsumgebung* ist ein softwaretechnisches Modell eines Arbeitsplatzes und seiner Umgebung (mehr hierzu in Abschnitt 3.4.1). Benutzer können ihre Arbeitsumgebung nach individuellem Bedarf einrichten. Beim nächsten Systemstart soll sich die Arbeitsumgebung so präsentieren, wie sie hinterlassen wurde. Es besteht auch die Möglichkeit, daß Benutzer in mehreren Arbeitsumgebungen arbeiten. Einige private Gegenstände (z.B. sein Kalender) stehen dem Benutzer in jeder Arbeitsumgebung zur Verfügung. Diese Gegenstände werden *Privatgebrauch* genannt (siehe [Roock & Wolf 98], Seite 33).

⇒ Das *Postversandsystem* erlaubt es Benutzern, anderen Benutzern softwaretechnische Gegenstände (insbesondere Materialien) zuzuschicken (siehe Abschnitt 3.7.2).

⇒ Innerhalb einer Organisation kann ein Benutzer unterschiedliche *anwendungsfachliche Rollen* innehaben. So kann zum Beispiel ein Mitarbeiter einer Bank gleichzeitig auch Kunde der Bank sein.

⇒ Wenn benutzerbezogene *Zugriffskontrollen* erfolgen sollen, müssen einem Benutzer Zugriffsrechte zugeordnet werden. Bei der in diesem Kapitel vorgestellten Schlüssel-Schloß-Metapher besitzt der Benutzer als Zugriffsrechte Schlüssel. Des weiteren tritt er als Verantwortlicher von Handlungen im Anwendungssystem in Erscheinung (vgl. Abschnitt 2.2.1).

In [dtv Lexikon 90] wird der Begriff Modell definiert als „Darstellung, die nur die als wichtig angesehenen Eigenschaften des Vorbildes ausdrückt“. Damit stellt sich die Frage, welche Eigenschaften das Modell des Benutzers beinhalten soll. Der Hintergrund, vor dem die Eigenschaften als wichtig oder weniger wichtig bewertet werden sollen, ist das Zugriffskontrollmodell, welches in dieser Arbeit beschrieben wird.

In den meisten Fällen wird offenbar eine fachliche Identität des Anwenders benötigt (vgl. Tabelle 2). Des weiteren sind Behälter für Schlüssel und Zertifikate sowie ein weiterer für Gegenstände für den Privatgebrauch vonnöten.

Ergebnis 11
Das anwendungsfachliche Modell eines Benutzers enthält seine fachliche Identität sowie je einen Behälter für Schlüssel und für Gegenstände des Privatgebrauchs.

Zur fachlichen Identität des Benutzers gehört zunächst der Name des Benutzers. Im Anwendungsbereich ist die Kombination von Vor- und Nachname meist hinreichend, um einen Anwender eindeutig zu bezeichnen. Anwender haben im Arbeitsalltag meist mit einer überschaubaren Anzahl von Kollegen zu tun, so daß die Wahrscheinlichkeit, daß zwei Kollegen denselben Namen haben, sehr gering ist. In größeren Organisationen kann es hingegen häufiger vor, das zwei Personen den gleichen Namen tragen. Zudem können sich Namen ändern (z.B. bei einer Heirat). In diesem Fall wird zusätzlich eine eindeutige Kennung, z.B. eine Personalnummer, zur eindeutigen Identifikation verwendet. Dies betrifft zumeist Anwendungsfälle, in denen kein persönlicher Kontakt stattfindet (z.B. Gehaltsabrechnung).

Kontext	Rolle des Benutzers	Realisierung im Modell des Benutzers
Zugriffskontrolle	• Besitzer von Schlüsseln • Verantwortlicher für Handlungen	⇒ Behälter für Schlüssel ⇒ fachliche Identität
Registratur-Konzept	• Erzeuger von Materialien • Bearbeiter von Materialien • Besitzer von Materialien • Entleiher von Materialien	⇒ fachliche Identität ⇒ fachliche Identität ⇒ fachliche Identität ⇒ fachliche Identität
Arbeitsumgebung	• Gestalter der Umgebung • Besitzer von Gegenständen für den Privatgebrauch	keine ⇒ Behälter für Schlüssel
Postversandsystem	• Empfänger von Post • Absender von Post	⇒ fachliche Identität ⇒ fachliche Identität
Anwendungsfachliche Rolle	• diverse (z.B. Kunde, Mitarbeiter)	keine

Tabelle 2: Rolle des Benutzers als Modell im Anwendungssystem

Im fachlichen Model des Benutzers nicht berücksichtigt wurden die Verwaltung der persönlichen Arbeitsumgebung sowie die Modellierung anwendungsfachlicher Rollen. Hierbei handelt es sich um eigenständige Konzepte, die gesondert betrachtet werden müssen. Der Schwerpunkt dieser Arbeit liegt auf der Beschreibung eines fachlichen Zugriffsmodells für WAM-Systeme, und für diesen Kontext ist das in Ergebnis 11 beschriebene fachliche Modell eines Benutzers ausreichend. Als offene Frage wird formuliert:

Offene Frage 2
Wie können anwendungsfachliche Rollen sowie die Verwaltung der persönlichen Arbeitsumgebung in das fachliche Modell eines Benutzers eingebunden werden?

3.2. Bisherige WAM-Konzepte zur Zugriffskontrolle

In WAM-Anwendungssystemen werden Anwendern die Werkzeuge und Materialien zur Verfügung gestellt, die sie für die Erledigung ihrer Aufgaben brauchen. Maxime war, daß alle Gegenstände, die ein Anwender erreichen kann, auch bearbeitet werden können.

Erste Einschränkungen fanden durch die „kooperative Materialzugriffskontrolle nach dem Bild von Original und Kopie“ statt (siehe [Weske, Wulf 94]). Hier wurde für verteilte WAM-Anwendungssysteme ein *Materialkoordinator* beschrieben, mit dessen Hilfe „konkurrierende Zugriffe von Werkzeugumgebungen auf gemeinsame Materialien“ innerhalb einer verteilten Umgebung koordiniert werden können ([Weske, Wulf 94], Seite 12). Das „Original und Ko-

pie“ Modell geht davon aus, daß von jedem Material zu einem Zeitpunkt immer genau ein Original und beliebig viele Kopien existieren. Ein Original wird nur von einem Benutzer zur Zeit bearbeitet. Andere Benutzer können während dieser Zeit nur Kopien des Originals im Zustand vor der aktuellen Bearbeitung erhalten. Verändernde Zugriffe können nur innerhalb einer lokalen Arbeitsplatzumgebung durchgeführt werden ([Wulf 95], Seite 66). Die Materialien werden daher aus einer verteilten in die lokale Arbeitsumgebung befördert. Eine wesentliche Aufgabe des Materialkoordinators ist die Autorisierung der Zugriffe und die Protokollierung der Materialbewegungen ([Weske, Wulf 94], Seite 13).

Damit existiert zwar eine fachlich motivierte Zugriffsmodellierung nach dem Bild von Originalen und Kopien, allerdings wird in den zitierten Arbeiten kein Mechanismus zur Zugriffskontrolle eingeführt oder beschrieben. Erste Vorschläge für eine konkrete Zugriffskontrolle mit Schlüsseln und Schlössern finden sich in [Roock & Wolf 98]. Ein entsprechendes Modell wird dort nicht näher beschrieben. Eine genaue Beschreibung der Schlüssel-Schloß-Metapher findet sich dort ebenfalls nicht. Sie ist der Gegenstand dieser Arbeit.

3.3. Vorstellung der Schlüssel-Schloß-Metapher

In diesem Abschnitt wird eine Interpretation der Schlüssel-Schloß-Metapher für den WAM-Kontext entwickelt. Mit Hilfe dieses Prinzips lassen sich Zugriffskontrollen in WAM durchführen. Ausgangspunkt ist das Leitbild „Arbeitsplatz für eigenverantwortliche Expertentätigkeit“. Dieses Leitbild harmoniert mit der Metapher besonders gut, da der eigenverantwortliche Umgang mit der Zugriffskontrolle im Vordergrund steht (siehe Kapitel 2.3.4).

Ausgehend von Schlüsseln und Schlössern in der Anwendungswelt wird die Metapher und ihr Verhältnis zu den WAM-Entwurfsmetaphern beschrieben. Darüber hinaus wird heraus gearbeitet, welche Änderungen sich zwischen der Metapher und ihrem Urbild ergeben.

3.3.1. Schlüssel und Schlösser in der Anwendungswelt[11]

Schlüssel und Schlösser werden seit mindestens 2500 Jahren von Menschen verwendet ([dtv Lexikon 90], Stichwort „Schloß“). Dabei wurden in früheren Jahrhunderten mechanische Schlüssel und Schlösser konstruiert und zum Teil auch magnetische Eigenschaften von Metallen benutzt. In den letzten Jahren halten immer mehr elektronische Schließsysteme (s.u.) Einzug in den Alltag (z.B. als Magnetkarten in Hotels oder als Chip im Autoschlüssel). Schlüssel und Schlösser sind jedermann aus dem täglichen Leben vertraut. In diesem Abschnitt soll ein Überblick darüber gegeben werden, wie Schlüssel und Schlösser aufgebaut sind und organisiert werden können.

Generelle Eigenschaften von Schlüsseln und Schlössern

Wenn reale Gegenstände im Anwendungsbereich vor unberechtigtem Zugriff geschützt werden sollen, werden hierzu meist Schlüssel und Schlösser verwendet. Daher wird betrachtet, wie sich Schlüssel und Schlösser und ihr Zusammenhang mit den Gegenständen des Anwendungsbereiches in Software übertragen lassen. Eine Menge von Schlössern und passenden Schlüsseln, in der die Schlüssel hierarchisch angeordnet sind, wird auch als Schließsystem bezeichnet. Ein reales Schließsystem, wie es zum Beispiel in einem Hotel vorkommen kann, wird in [Beis, Vorwerk 97] beschrieben:

> **Schließsystem eines Hotels:**
> Bei herkömmlichen, mechanischen Schlüsseln handelt es sich um Sicherheits-Schließsysteme mit mehreren Ebenen. Es gibt Einzel-, Gruppen- und Generalschlüssel. Je nach Form des Schlüssels werden bestimmte Metallstifte im Schloßzylinder bewegt, so daß eine Tür geöffnet werden kann. Problematisch wird die Verwendung des herkömmlichen Schlüsselsystems bei Verlust eines Schlüssels. Hier muß das komplette Schloß oder - im schlimmsten Fall - das Schließsystem

[11] Unter Anwendungswelt wird die Welt außerhalb des Anwendungssystems verstanden. Anwendungssystem und Anwendungswelt können sich durchaus gegenseitig beeinflussen.

erneuert werden, was mit hohen Kosten verbunden ist. Wird hingegen nur der Schlüssel ersetzt, ist der Zugang fremder Personen zu dem entsprechenden Raum weiterhin durch den verlorengegangenen Schlüssel möglich.

Neben Türen lassen sich auch andere Gegenstände abschließen. Bei diesen Gegenständen handelt es sich meist um Behälter (zum Beispiel ein Schließfach). Weiterhin werden Schlösser zum Teil an Automaten vorgefunden. Schlüsselschalter (Schalter, die durch einen Schlüssel aktiviert oder deaktiviert werden) finden sich z.B. in Kraftfahrzeugen. Viele Bedienelemente (z.B. Blinkerschalter, Gebläseregler) funktionieren erst, wenn der Kfz-Schlüssel im Schloß auf eine bestimmte Position gedreht wurde. Bei einigen Fahrzeugtypen verändert sich in diesem Fall auch die Funktionalität des Blinkerschalters: im Ruhezustand wird das Standlicht und im Betrieb das Blinklicht aktiviert. Schlösser können auch an Ketten befestigt sein, mit denen Gegenstände angekettet werden. Nur der Besitzer eines passenden Schlüssels kann den Gegenstand dann aus seiner Umgebung entfernen.

Herstellung von Schlüsseln und Schlössern

Ein weiterer Aspekt ist die Produktion von Schlüsseln und Schlössern. Reale Schlüssel und Schlösser werden normalerweise von einem Hersteller bezogen, und Anwender müssen sich nicht um ihre Herstellung kümmern, wohl aber um ihre Beschaffung.

Einfache Schlüssel können jederzeit nachgemacht werden, indem der Anwender sie zum Beispiel von einem Schlüsseldienst duplizieren läßt. Schlüssel und Schlösser werden nach ihrer Erzeugung nicht mehr verändert. Statt dessen werden bei Bedarf alte Schlüssel / Schlösser durch neue ersetzt. Einfache Schlüssel und Schlösser werden aus Kostengründen in größeren Auflagen produziert. Einige Anwender erhalten baugleiche Schlüssel. Die Sicherheit der Schlösser beruht dann darauf, daß es sehr unwahrscheinlich ist, neben dem eigenen ein weiteres passendes Schloß zu finden.

Im Kontext einer Organisation (z.B. Firma oder Behörde) werden meist Sicherheitsschlüssel verwendet, die mit einem Zertifikat ausgeliefert werden. Nur wer dieses Zertifikat präsentieren kann, bekommt auf legalem Weg eine Kopie eines solchen Schlüssels. Zum Teil wird vom Hersteller des Zertifikates auch dessen Eigentümer registriert. Wird ein neuer Schlüssel benötigt, muß der Eigentümer zusätzlich seine Identität nachweisen.

Organisation von Schlüsseln und Schlössern

Schlüssel und Schlösser können in einem Schließsystem hierarchisch angeordnet werden. Ein hierarchisch übergeordneter Schlüssel kann alle Schlösser öffnen, die ein untergeordneter Schlüssel öffnen kann. Die hierarchischen Beziehungen zwischen Schlüsseln werden im Rahmen dieser Arbeit als *Sub-Schlüssel* bzw. *Super-Schlüssel* bezeichnet:

> **Definition 19: Sub-/ Super-Schlüssel**
> Stehen zwei Schlüssel in einer hierarchischen Beziehung zueinander, heißt der hierarchisch übergeordnete Schlüssel Super-Schlüssel und der hierarchisch untergeordnete Schlüssel Sub-Schlüssel.

Ein Super-Schlüssel kann neben den Schlössern seiner Sub-Schlüssel gegebenenfalls noch weitere Schlösser öffnen. Eine einfache, hierarchische Organisation von Schlüsseln und Schlössern wird im folgenden exemplarisch dargestellt.

Abbildung 10 stellt ein Bürogebäude dar. Insgesamt gibt es hier zwei Etagen mit je drei Büros. Das Haus verfügt über eine Eingangstür und auf jeder Etage über eine Etagentür. Jeder Mitarbeiter benötigt nur einen Schlüssel, um Eingangstür, Etagentür und Bürotür zu öffnen. Darüber hinaus hat das Reinigungspersonal Schlüssel für je eine gesamte Etage sowie der Hausmeister einen Generalschlüssel (siehe Tabelle 3).

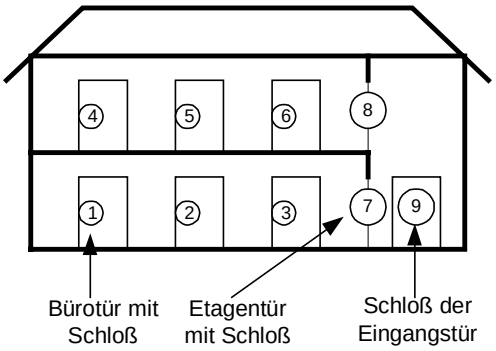


Abbildung 10: Schlösser in einem Bürogebäude

Schlüssel-Besitzer	Schlüssel-Typ	Türen
Mitarbeiter 1	Zimmer-Schlüssel	1,7,9
Reinigungs-personal unten	Etagen-Schlüssel	1,2,3,7,9
Hausmeister	General-Schlüssel	1,...,9

Tabelle 3: Beispiel für die Schlüsselorganisation eines Büros

Die entsprechende Schlüsselhierarchie zeigt Abbildung 11. Die Schlüssel werden innerhalb einer Organisation den entsprechenden Mitarbeitern ausgehändigt. Da Schlüssel im Besitz von Anwendern sind, können sie auch von ihnen weitergegeben werden. Anwender tragen eine Mitverantwortung für die Sicherheit des Schließsystems. Es liegt in ihrem Ermessen, ob und an wen sie Schlüssel weitergeben. Die Rückgabe von Schlüsseln kann vor Gericht erzwungen werden.

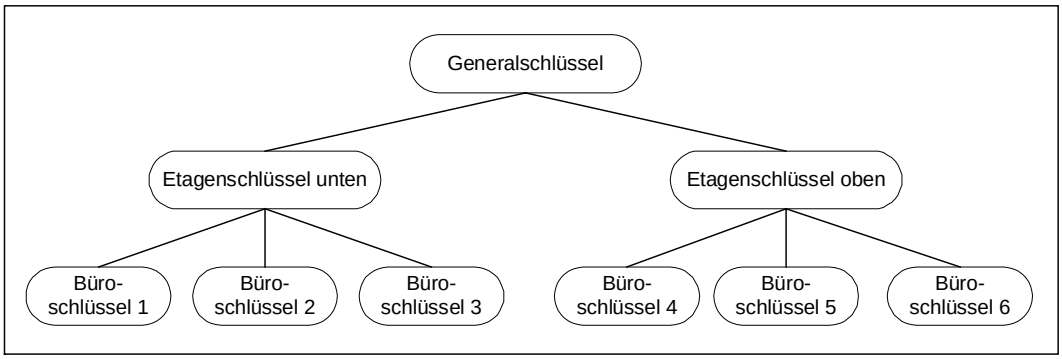


Abbildung 11: Schlüsselhierarchie

3.3.2. Metaphern im WAM-Ansatz

Laut [dtv Lexikon 90] ist eine Metapher eine „Redewendung, in der statt der eigentlichen Bezeichnung eine uneigentliche oder übertragene gebraucht wird“. Danach diene die Metapher dank ihrer „Bildhaftigkeit“ besonders zur „Veranschaulichung von Geistigem“.

Diese „Bildhaftigkeit“ wird im WAM-Kontext bewußt eingesetzt, „um Leitbilder anschaulicher und damit ihre Wirkung deutlicher zu machen“. Metaphern werden zur „Gestaltung von Softwaresystemen“ verwendet und daher auch als „Entwurfsmetaphern“ bezeichnet (siehe Abschnitt 2.3.1). Sie sollen den am Softwareentwicklungsprozeß Beteiligten die „Handhabung und Funktionalität“ des Systems verdeutlichen, indem sie „die Vorstellung und Kommunikation über das, was fachlich analysiert, modelliert und technisch realisiert werden soll“, bildlich darstellt (siehe [Züllighoven et al. 98], Seite 79).

Das Ziel der Verwendung einer Metapher ist, „bestimmte Eigenschaften oder Gesichtspunkte am ursprünglichen Ausdruck“ hervorzuheben ([Züllighoven et al. 98], Seite 79). Eine Entwurfsmetapher muß gleichzeitig Abstraktionen von dem „Urbild“ vornehmen, um sinnvoll eingesetzt werden zu können. Ansonsten würde es sich nicht um eine Metapher, sondern um ein konkretes Modell eines Gegenstandes oder Konzeptes handeln.

3.3.3. Die Schlüssel-Schloß-Metapher für WAM-Systeme

Schlüssel und Schlösser haben Materialcharakter. Dies wird aus der Definition des Begriffes *Material* deutlich:

Definition 20: Material
Gegenstand, der im Rahmen einer Aufgabe Teil des Arbeitsergebnisses wird. Materialien werden durch Werkzeuge und Automaten bearbeitet und verkörpern fachliche Konzepte. Sie müssen für die Bearbeitung geeignet sein. (siehe [Züllighoven et al. 98])

Schlüssel und Schlösser werden allerdings nach ihrer Erzeugung nicht mehr bearbeitet und werden auch nicht „Teil eines Arbeitsergebnisses“. Sie dienen einem speziellen Zweck und werden daher als spezielle Metaphern definiert:

Definition 21: Schloß
Ein Schloß ist ein Gegenstand, der entweder verschlossen oder unverschlossen ist. Es ist Teil eines anderen Gegenstandes oder wird an diesem angebracht, um ihn verschließen zu können. Verschlossene Gegenstände können gar nicht oder nur eingeschränkt verwendet bzw. geöffnet werden. Schlösser sind die fachlichen Zugriffskontrollen in WAM-Systemen.

Definition 22: Schlüssel
Ein Schlüssel ist ein Gegenstand, mit dem man Schlösser auf- und abschließen kann. Schlüssel stellen das fachliche Modell für Zugriffsrechte in WAM-Systemen dar. Der Besitzer eines Schlüssels ist ein Anwender.

Offensichtlich stehen Schlösser in engem Zusammenhang mit anderen Gegenständen. Dieser Zusammenhang wird in Abschnitt 3.4 auf der Ebene von Entwurfsmetaphern untersucht.

Schlüssel und Schlösser werden von einem fachlichen Service[12] erzeugt. Von diesem erhält ein Anwender ein neues Schloß oder einen neuen Schlüssel gegen Vorlage eines Zertifikates. Auf dem Zertifikat steht der Name des Eigentümers eines Zertifikates. Nur dem Eigentümer des Zertifikates ist es möglich, Schlüssel und Schlösser hiermit anzufertigen. Sollte ein anderer Anwender in den Besitz des Zertifikates gelangen, kann er damit nichts anfangen, da er nicht der Eigentümer des Zertifikates ist. Damit stellt sich unmittelbar die Frage nach dem Unterschied zwischen *Eigentum* und *Besitz*:

Definition 23: Eigentum
Eigentum ist das unmittelbar dingliche Recht einer Person auf Besitz, Nutzung und Verfügung über eine Sache unter Ausschluß aller anderen Personen. ([Encarta 99])

Eigentümer eines Gegenstandes ist also derjenige, dem der Gegenstand rechtlich zugeordnet ist. Der Eigentümer darf mit einem Gegenstand beliebig verfahren.

Definition 24: Besitz
Besitz drückt die tatsächliche Herrschaft einer Person über einen Gegenstand aus. (nach [Encarta 99])

Der Besitzer eines Gegenstandes kann ihn nach eigenem Interesse gebrauchen und nutzen. Ein Beispiel für den Unterschied zwischen Eigentum und Besitz ist ein Koffer: kauft man sich einen Koffer in einem Geschäft, ist man gleichzeitig Eigentümer und Besitzer. Wird einem der Koffer anschließend gestohlen, bleibt man zwar der Eigentümer, die tatsächliche Herrschaft und damit den Besitz an dem Koffer hat aber der Dieb (nach [Encarta 99]).

Nur der Eigentümer eines Zertifikates ist berechtigt, hierfür neue Schlüssel und Schlösser anzufertigen. Vor der Erzeugung wird überprüft, ob der Besitzer des Zertifikates auch dessen Eigentümer ist. Genau die Schlüssel, die für ein Zertifikat erzeugt wurden, passen zu allen Schlössern, die für dasselbe Zertifikat erzeugt wurden.

[12] Die Definition *fachlicher Services* erfolgt in Abschnitt 3.4.6.

Definition 25: Zertifikat
Ein Zertifikat ist ein Gegenstand, der einen Anwender zur Erzeugung eines Schlosses oder eines Schlüssels berechtigt. Ein Anwender ist als Eigentümer auf dem Zertifikat eingetragen. Der Eigentümer eines Zertifikates ist gleichzeitig Eigentümer aller Schlüssel und Schlösser, die für dieses Zertifikat erzeugt wurden. Zwei Zertifikate sind niemals identisch.

Wie in Abschnitt 3.3.1 erwähnt, werden Schlüssel häufig in Schließsystemen organisiert. Die Schlüssel und Schlösser stehen dann in einer hierarchischen Beziehung zueinander. Diese Hierarchie soll auch durch Zertifikate ausgedrückt werden. Der Besitzer eines hierarchisch übergeordneten Zertifikates kann hierarchisch untergeordnete Zertifikate anfertigen lassen. Diese Zertifikate gehören ebenfalls ihm.

Definition 26: Sub-/Super-Zertifikate
Ein hierarchisch übergeordnetes Zertifikat heißt Super-Zertifikat, und ein hierarchisch untergeordnetes Zertifikat heißt Sub-Zertifikat.

Analog heißen die für ein Zertifikat erzeugten Schlüssel dann Sub- bzw. Super-Schlüssel. Eine entsprechende Benennung für Schlösser wird nicht vorgenommen. Super-Schlüssel können alle Schlösser öffnen, die durch ihre Sub-Schlüssel geöffnet werden können.

Abbildung 10 zeigt ein hierarchisches Schließsystem. Für dieses Schließsystem stellt Abbildung 12 in Ausschnitten dar, in welcher Beziehung Zertifikate, Schlösser und Schlüssel zueinander stehen.

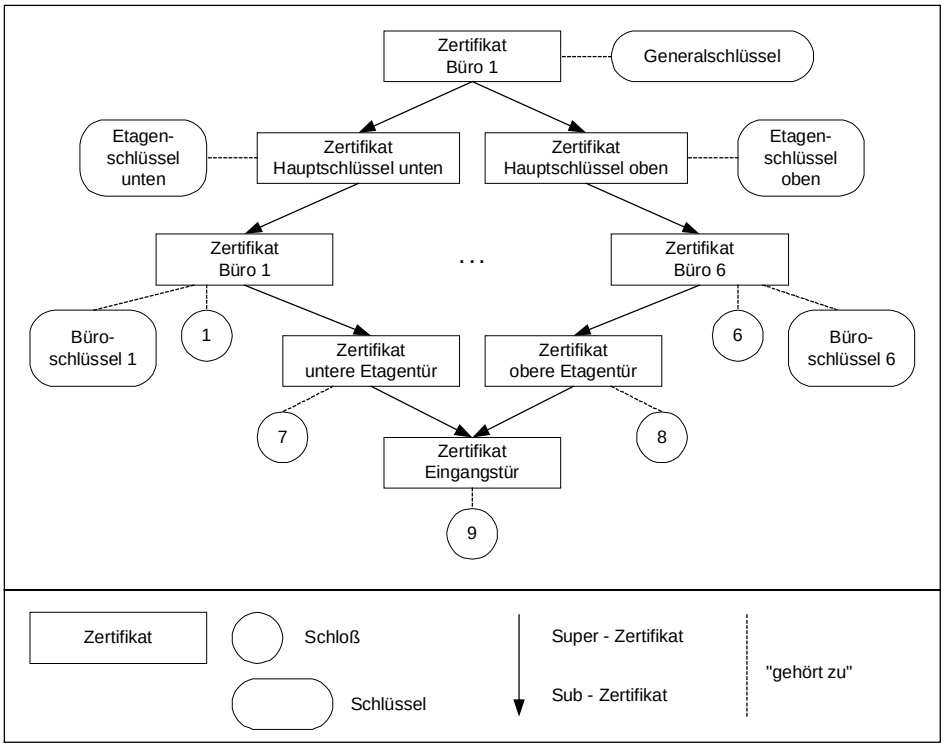


Abbildung 12: Schematische Darstellung eines Schließsystems

Der Besitzer eines Zertifikates kann Schlüssel erzeugen und sie anderen Anwendern zur Verfügung stellen. Diese sind dann zwar Besitzer, aber nicht Eigentümer des neuen Schlüssels. Sie können entsprechende Schlösser mit dem Schlüssel öffnen, aber diese Schlösser nicht entfernen. Auf diese Weise kann ein Administrator eines Anwendungssystems Zugriffskontrollen erzwingen, die von Mitarbeitern nicht umgangen werden können. Andererseits haben Anwender die Möglichkeit, eigene Zertifikate zu erzeugen und zum Beispiel private Behälter

mit entsprechenden Schlössern zu verschließen. Die Kontrolle über die Zugriffskontrolle liegt dann nicht in der Hand des Administrators, sondern in der des einzelnen Anwenders.

Schlüssel, Schlösser und Zertifikate sind innerhalb eines Anwendungssystems global gültig. Wenn ein Schlüssel einmal zu einem Schloß paßt, kann er es immer öffnen. Eine temporäre Gültigkeit oder eine ereignisgesteuerte Begrenzung der Gültigkeit ist, wie bei realen Schlüsseln, nicht vorgesehen.

Diese Eigenschaften von Schlüsseln wären durchaus sinnvolle Ergänzungen am Modell eines Schlüssels. Auch in der Anwendungswelt findet man sogenannte Zeitschlösser, die sich nicht zu jedem Zeitpunkt öffnen lassen. Allerdings ergeben sich hieraus deutliche Abweichungen von der ursprünglichen Metapher. Diese Abweichungen führen erfahrungsgemäß zu Seiteneffekten, die im Rahmen dieser Arbeit nicht behandelt werden können.

3.3.4. Das Benutzungsmodell

Ein Benutzungsmodell ist „ein Modell darüber, wie Anwendungssoftware bei der Erledigung der anstehenden Aufgaben im jeweiligen Einsatzkontext benutzt werden kann“ [Gryczan et al. 00]. Das hier vorgestellte Benutzungsmodell stellt den grundlegenden Umgang mit Schlüsseln und Schlössern dar.

Der grundsätzliche Umgang mit Schlüsseln und Schlössern

Schlüssel befinden sich im Besitz von Benutzern. Ein Schloß kann mit einem Schlüssel explizit auf- beziehungsweise abgeschlossen werden. Beim expliziten Schließen versetzt der Anwender es bewußt vom Zustand „verschlossen“ in den Zustand „unverschlossen“ (bzw. von „unverschlossen“ zu „verschlossen“).

> *Vision: Explizites Schließen auf dem Desktop*
> Der Anwender möchte einen verschlossenen Behälter öffnen. Er zieht das Symbol seines Schlüsselbundes auf den Behälter. Mit einem passenden Schlüssel vom Schlüsselbund wird das Schloß geöffnet. Der Behälter ist nun offen und ein Auflister-Werkzeug wird gestartet. Nachdem der Anwender seine Aufgabe ausgeführt hat, schließt er das Auflister-Werkzeug und damit auch den Behälter. Dieser ist nun noch nicht abgeschlossen. Um den Behälter abzuschließen, benutzt der Anwender wieder seinen Schlüsselbund (s.o.). Schließt er den Behälter nicht ab, bleibt er unverschlossen, bis ein Anwender ihn explizit abschließt.

An einem Schloß ist erkennbar, ob es verschlossen ist oder nicht. Zum Teil kann auf der Ebene der Metapher entschieden werden, wie sich ein verschlossenes Schloß auswirkt (siehe z.B. Abschnitt 3.4.1). Häufig hängt dies jedoch vom konkreten Gegenstand ab. Zum Beispiel können in einen verschlossenen Briefkasten immer noch Briefe hinein geworfen werden, während dies bei einem verschlossenen Tresor nicht möglich ist.

Der Anwender sollte dem Gegenstand ansehen oder sich zumindest darüber informieren können, ob eine gewünschte Aktion mit einem verschlossenen Gegenstand möglich ist. Auf jeden Fall erfolgt eine Benachrichtigung des Anwenders, wenn er eine Aktion ausführt und an der Zugriffskontrolle scheitert.

Die obige Vision zeigt durch den Sprachgebrauch im Übrigen auch, wie sehr Schloß und Gegenstand als eine Einheit gesehen werden können. Offensichtlich sind Schlösser häufig elementarer Bestandteil von Gegenständen, insbesondere von Behältern.

Erzeugung von Schlüsseln, Schlössern und Zertifikaten

Schlüssel, Schlösser und Zertifikate werden von einem fachlichen Service erzeugt (siehe Abschnitt 3.3.3). Soll ein Sub-Zertifikat erzeugt werden, benötigt der Anwender Zertifikate, welche dann zu Super-Zertifikaten des neue erzeugten Sub-Zertifikates werden. Hierarchisch am höchsten angesiedelt ist das Zertifikat für den Generalschlüssel.

Der Eigentümer eines Zertifikates kann bei dessen Vorlage auch Schlüssel und Schlösser für dieses Zertifikat erzeugen. Alle für ein Zertifikat erzeugten Schlüsseln passen zu allen für dieses Zertifikat erzeugten Schlösser sowie zu allen Schlösser entsprechender Sub-Zertifikate.

Der fachliche Service zum Erzeugen von Schlüsseln und Schlössern ist bei der ersten Inbetriebnahme des Systems abgeschlossen. Für die Systemkonfiguration ist es notwendig, daß zu diesem Zeitpunkt bereits ein registrierter Benutzer, der Administrator, existiert. Dieser verfügt über den Generalschlüssel samt zugehörigem Zertifikat.

Bewegen von Schlüsseln, Schlössern und Zertifikaten
Schlüssel, Schlösser und Zertifikate können aufgrund ihrer Gegenständlichkeit prinzipiell bewegt werden. Schlüssel werden von ihren Besitzern normalerweise herumgetragen und Schlösser können ausgebaut oder entfernt und an anderer Stelle wieder verwendet werden.

Zertifikate gehören normalerweise zum Privatgebrauch eines Anwenders (vgl. Abschnitt 3.1). Ein Anwender kann sich entscheiden, die Eigentümerschaft über ein Zertifikat einem anderen Anwender zu übertragen. Hierzu wendet er sich mit dem Zertifikat an den Service, der Zertifikate erzeugt, und gibt diesem die Änderung der Eigentumsverhältnisse bekannt. Der Service trägt den neuen Eigentümer in das Zertifikat ein und leitet es an den neuen Eigentümer weiter.

Weder Schlüssel noch Schloß können sich zu einem Zeitpunkt an mehreren Stellen in einem System befinden (Einheit von Zeit und Raum). Es kann allerdings Kopien geben.[13] Dieser Aspekt bei der Betrachtung von Schlüsseln und Schlössern ist wesentlich, da Schlüssel auch weitergegeben werden können, wenn es die Situation erfordert. Der Besitzer ist somit in der Lage, seine Zugriffsrechte an einen anderen Anwender zu übertragen. Dieser Fall kann zum Beispiel eintreten, wenn der Anwender während seiner Abwesenheit eine Vertretung braucht. Der Vertreter erhält dann einen Schlüssel für die Arbeitsumgebung, nicht aber zum Beispiel für einen Behälter, in dem private Dokumente liegen.

Bei verantwortungsvollem Umgang erweist sich diese Handhabung als Vorteil, da Zugriffskontrollen damit eine häufig verlangte Flexibilität erlangen. Allerdings entstehen Probleme, wenn man sich nicht auf das verantwortungsvolle Handeln aller Mitarbeiter verlassen kann oder will. Daher sollte die Weitergabe von Schlüsseln und Zertifikaten beschränkt werden können. Diese Diskussion wird im Zusammenhang mit der Betrachtung anderer Leitbilder nochmals aufgegriffen (siehe Abschnitt 3.6).

Verwaltung von Schlüsseln, Schlössern und Zertifikaten
Die Verwaltung von Schlüsseln und Schlössern erfolgt prinzipiell durch den Besitzer des jeweiligen Gegenstandes, bei Zertifikaten durch den Eigentümer. Ein Administrator kann entscheiden, daß andere Anwender das Werkzeug zum Erzeugen von Zertifikaten nicht benutzen können. Hierzu wird das Werkzeug selbst abgeschlossen und nur der Administrator verfügt über einen Schlüssel. Dann ist es nur ihm möglich, Schlüssel und Schlösser zu erzeugen, so daß er die volle Kontrolle über alle Zugriffskontrollen im Anwendungssystem behält.

Soll ein Anwender ein Zugriffsrecht bekommen, muß ihm der Eigentümer des Schlosses einen Schlüssel zukommen lassen. Soll ein Anwender ein Zugriffsrecht verlieren, muß er einen Schlüssel freiwillig wieder abgeben. Ist dies nicht möglich, muß ein neues Schloß angebracht werden, zu dem der entsprechende Anwender dann keinen Schlüssel mehr erhält.

3.3.5. Sicherheitsrisiken

Es soll nicht vernachlässigt werden, daß auch bei softwarebasierten Schlüsseln Probleme auftreten können. Wie bereits erwähnt, wird im Rahmen dieser Arbeit auf technische Risiken nicht näher eingegangen, die durch den Einsatz einer Programmiersprache, eines konkreten Persistenzmediums o.ä. entstehen. Allerdings gibt es auch auf fachlicher Ebene Sicherheitsrisiken. Ein Problem entsteht, wenn ein Anwender einen Schlüssel erhält, diesen aber nicht

[13] Zu „Einheit von Zeit und Raum“ sowie zu „Original-Kopie“ – Diskussion siehe [Züllighoven et al. 98] (Seite 431ff)

wieder zurückgeben möchte. Dann ist, wie beim Verlust eines mechanischen Schlüssels, der Austausch von Teilen des Schließsystems notwendig.

Können Anwender Schlüssel und Zertifikate selbständig nachmachen und untereinander austauschen, liegt die Kontrolle über Zugriffskontrollen nicht mehr in der Hand des Systemadministrators. Zudem kann der Verantwortliche einer Aktion nicht mehr zweifelsfrei ermittelt werden, denn der verwendete Schlüssel könnte zum fraglichen Zeitpunkt jedem Anwender gehört haben. Es wird daher empfohlen, den Zugriff auf besonders schutzbedürftige Systemkomponenten zu protokollieren (vgl. Abschnitt 4.5) und die Möglichkeit zur Weitergabe von Schlüsseln und Zertifikaten einzuschränken (z.B. auf private Schlüssel). Diese Möglichkeiten werden im folgenden nicht weiter erörtert. Als offene Frage verbleibt:

Offene Frage 3
Wie kann die Weitergabe von Schlüssel und Zertifikate durch Anwender verhindert werden?

3.4. Bezug zu den anderen WAM - Metaphern

Wenn Schlüssel und Schlösser als Metaphern eingeführt werden, stellt sich die Frage nach ihrem Zusammenhang mit den WAM-Entwurfsmetaphern. Dieser Zusammenhang soll im folgenden für die verschiedenen Entwurfsmetaphern erörtert werden. Dabei wird von einer Arbeitssituation ausgegangen, in der ein Anwender lokal innerhalb einer Softwareumgebung arbeitet. Für diesen Fall läßt sich die Schlüssel-Schloß-Metapher besonders gut diskutieren. Auf andere Leitbilder wird später eingegangen.

3.4.1. Arbeitsumgebung - Arbeitsplatz - Desktop

Qualifizierte fachliche Arbeit wird immer an einem Ort verrichtet ([Gryczan 96], Seite 152). Normalerweise ist dieser Ort mit den Gegenständen ausgestattet, die bei der Arbeit benötigt werden. Innerhalb einer Arbeitsumgebung sollen Gegenstände „ohne zeitliche Verzögerung" bearbeitet werden können, und die Effekte sollen „unmittelbar sichtbar" sein ([Züllighoven et al. 98]). Die Entwurfsmetapher *Arbeitsumgebung* ist dort definiert als:

Definition 27: Arbeitsumgebung
Ort, wo Werkzeuge, Automaten und Materialien, die bei der Erledigung einer Aufgabe griffbereit sein müssen, fachlich motiviert angeordnet sind. Die eigentliche Arbeit findet am Arbeitsplatz statt. Zur Umgebung gehören noch die Orte, die unmittelbar zugänglich sind.

Die Arbeitsumgebung stellt einen eigenen Bereich innerhalb eines Anwendungssystems dar, der vor Zugriff von außen geschützt ist ([Züllighoven et al. 98], Seite 178). Von der Arbeitsumgebung nimmt der Anwender normalerweise nur seinen Arbeitsplatz wahr, der zum Beispiel durch einen Desktop dargestellt werden kann (siehe hierzu [Lippert 99])[14]. In WAM-Systemen können Anwender einen Desktop nur über die grafische Benutzungsschnittstelle eines Arbeitsplatzrechners verwenden. Diese Arbeitsplatzrechner stehen oft an Stellen, die potentiell vielen Anwendern zugänglich sind. Daher muß der persönliche Desktop eines Anwenders zusätzlich geschützt werden. Dies geschieht durch eine Zugangskontrolle:

Vision: Start eines privaten Arbeitsplatzes
Der Anwender startet die WAM-Benutzeroberfläche. Es erscheint ein Login-Fenster, mit der sich der Anwender authentifiziert. Anschließend werden ihm alle Desktops zur Auswahl angeboten, zu denen er einen Schlüssel besitzt. Er entscheidet sich für seinen privaten Desktop. Dieser wird nun aufgeschlossen und dargestellt. Der Anwender sieht den Desktop so, wie er ihn hinterlassen hat.

[14] Dabei ist zu beachten, daß ein Desktop die Arbeitsumgebung visualisiert, aber nicht mit ihr gleichzusetzen ist. Im allgemeinen präsentiert ein Desktop einen Teil der Arbeitsumgebung.

Wie oben erwähnt, kann der Anwender von seiner Arbeitsumgebung nur den Arbeitsplatz wahrnehmen, der wiederum gegebenenfalls durch einen Desktop dargestellt wird. Wenn Zugriffsschutz benötigt wird, ist es sinnvoll, diesen Arbeitsplatz abzuschließen. In der Anwendungswelt entspricht dies dem Abschließen eines Einzelbüros. Ist ein Arbeitsplatz abgeschlossen, kann nicht mehr an ihm gearbeitet werden. Dies entspricht einer Abmeldung vom Anwendungssystem. Erst nach einer erneuten Zugangskontrolle kann der Arbeitsplatz wieder benutzt werden.

In den bisherigen Betrachtungen zur Umgebung war ein Zugriffsschutz für Dinge innerhalb einer Arbeitsumgebung nicht vorgesehen ([Züllighoven et al. 98], Seite 182). In Abhängigkeit vom Anwendungskontext kann diese Sichtweise auch weiterhin Sinn machen. Allerdings gibt es auch Anwendungsfälle, in denen innerhalb einer Arbeitsumgebung Zugriffsschutz erfolgen muß. Beispiele sind Gruppenarbeitsplätze (siehe Abschnitt 3.6.2).

> **Ergebnis 12**
> Ein Arbeitsplatz kann abgeschlossen werden. Er ist dann nicht mehr benutzbar und wird auch nicht durch einen Desktop dargestellt. Soll ein abgeschlossener Arbeitsplatz wieder benutzt werden, muß eine erneute Zugangskontrolle stattfinden. Innerhalb einer Arbeitsumgebung können weitere Zugriffskontrollen stattfinden.

Die Umgebung verfügt nach der Anmeldung über ein fachliches Modell des Anwenders (siehe Abschnitt 3.1). Bei diesem können „Werkzeuge bei ihrer Erzeugung den Berechtigungsstatus eines Anwenders abfragen“ ([Züllighoven et al. 98], Seite 183), was hier natürlich mit Schlüsseln realisiert werden soll.

3.4.2. Behälter

In praktisch jedem Anwendungsbereich werden Behälter benötigt - zum Beispiel Aktenschränke oder Postkörbe. Üblicherweise sind diese Behälter nicht einfach eine Ablage für gerade nicht benötigte Gegenstände, sondern haben auch eine fachliche Funktionalität (z.B. als Archive oder Postfächer). Daher sind *Behälter* in [Züllighoven et al. 98] als eigene Entwurfsmetapher formuliert worden:

> **Definition 28: Behälter**
> In Behältern können Materialien aufbewahrt, gesammelt und angeordnet werden. Ein Behälter kann die aufbewahrten Materialien verwalten und über die Sammlung fachliche Auskünfte geben. Materialien können mit ihren Behältern an verschiedene Orte transportiert werden.

Behälter sind neben Türen die Gegenstände, die in der Realität am häufigsten mit Schlössern versehen werden und Anwendern ist das Verschließen von Behältern durchaus geläufig. Daher handelt es sich auch um die Entwurfsmetapher, die mit der Schlüssel-Schloß-Metapher am besten harmoniert. Die Beispiele mit dem Briefkasten und dem Tresor aus Abschnitt 3.3.4 zeigen, daß der Zustand „abgeschlossen“ nicht den gesamten fachlichen Umgang mit dem Behälter bestimmt. Während ein Tresor weder seinen Inhalt noch Informationen hierüber preisgibt, können in einen verschlossenen Briefkasten noch Gegenstände hineingelangen. Eine verschlossene Glasvitrine wiederum erlaubt die Sichtung und Sondierung, nicht aber eine Veränderung des Inhaltes. Wie sich verschlossene Behälter verhalten, kann offensichtlich nicht auf allgemeiner Ebene, sondern nur im konkreten Fall entschieden werden.

> **Ergebnis 13**
> Behälter können mit Schlössern versehen werden. Der fachliche Umgang eines Behälters kann nur im konkreten Fall (z.B. für einen Briefkasten oder einen Tresor) beschrieben werden.

3.4.3. Material

Betrachtet man die Materialien (z.B. einen Kreditvertrag) im Anwendungsbereich, so stellt man fest, daß diese selbst nicht abgeschlossen werden können. Vielmehr werden sie an speziellen Orten aufbewahrt, wenn sie vor unberechtigtem Zugriff geschützt werden sollen (so zum Beispiel Kundendaten in einem verschließbaren Register oder Verträge in einem Tresor). Dies soll auch bei der Modellierung im Anwendungssystem erhalten bleiben. Dort sind die entsprechenden Behälter vorhanden, wenn das Anwendungssystem richtig modelliert wurde.

> **Ergebnis 14**
> Materialien werden nicht mit Schlössern versehen.

Bei der Modellierung von Materialien muß berücksichtigt werden, daß einzelne Teile von Materialien nicht mit Zugriffsschutz versehen werden können. Es kann allerdings fachlich motiviert sein, daß bestimmte Teile eines Materials oder auch das gesamte Material in einem Arbeitszusammenhang nicht bearbeitet, sondern nur betrachtet werden können („read only"-Zugriff). Dies läßt sich jedoch nicht am Material modellieren. Materialien können von Anwendern nur durch Werkzeuge betrachtet beziehungsweise bearbeitet werden. Entsprechende Zugriffskontrollen sind demnach beim Werkzeug anzusiedeln.

> **Ergebnis 15**
> Partieller Schutz von Materialien (z.B. nur-lesender Zugriffe) kann nur durch die entsprechende Absicherung von Werkzeugen realisiert werden.

3.4.4. Werkzeug

Zur Bearbeitung von Materialien werden Werkzeuge verwendet. Ein Beispiel aus dem Bankenbereich ist eine Überweisung (das Material), die von einem Überweisungswerkzeug ausgefüllt wird. Im Vergleich zu Materialien und Behältern, die man meist als Gegenstände oder Konzepte im Anwendungsbereich findet, ist die Modellierung von guten Werkzeugen schwer. Die Entwurfsmetapher *Werkzeug* ist in [Züllighoven et al. 98] wie folgt definiert:

> **Definition 29: Werkzeug**
> Gegenstand, mit dem Menschen im Rahmen einer Aufgabe Materialien verändern oder sondieren können. Werkzeuge eignen sich meist für verschiedene Zwecke und für die Arbeit an unterschiedlichen Materialien. Sie müssen geeignet gehandhabt werden.

Der Sinn eines Werkzeugs ist demnach die Bearbeitung bzw. Sondierung eines Materials. Werkzeuge können aber auch auf Behälter einwirken und ihren Inhalt anzeigen bzw. verändern. Aufgrund von gescheiterten Zugriffskontrollen können Werkzeuge entweder gar nicht oder nicht in vollem Umfang verwendet werden. Zusätzlich ist es möglich und sinnvoll, die Benutzung eines Werkzeugs durch einen Anwender in einem Protokoll zu dokumentieren.

Werkzeuge können (wie Materialien) in unterschiedlichen Behältern aufbewahrt und damit weggeschlossen werden. So hat man prinzipiell die Möglichkeit, Werkzeuge nur bestimmten Anwendern oder Benutzergruppen zugänglich zu machen. Motiviert werden kann dies zum Beispiel durch eine mangelnde fachliche Qualifikation einzelner Anwender im Umgang mit dem Werkzeug.

Diese Sichtweise auf die Sicherung von Werkzeugen entspricht der eigentlichen Werkzeugmetapher und auch der Anwendungswelt am genausten. Wenn ein Werkzeug optimal für die Bearbeitung eines Materials durch Experten bei klarer Aufgabenstellung ausgelegt sein soll, kann es nicht gleichzeitig mit diversen Schutzmechanismen versehen werden. Wenn ein nicht hinreichend qualifizierter Anwender in eine Umgebung gelangen kann, in der solche Werkzeuge verfügbar sind, müssen die entsprechenden Werkzeuge in abschließbaren Behältern gelagert werden. Nur hinreichend qualifizierte Anwender erhalten dann einen Schlüssel.

Eine andere Möglichkeit ist, Werkzeuge direkt mit Schlössern zu versehen. Anwender können die Werkzeuge dann nur starten, wenn das Schloß aufgeschlossen ist. Der Schutz durch einen verschlossenen Behälter, in dem sich das Werkzeug befindet, ist nicht ausreichend. Schließlich können Anwender das Werkzeug einfach aus dem Behälter herausnehmen und es anderen Anwendern zur Verfügung stellen. Eine Sicherheitspolitik könnte auf diese Weise einfach unterlaufen werden. Zudem ist es auch komfortabler, wenn ein Werkzeug nicht immer in einem verschlossenen Behälter sondern z.B. auf dem Desktop liegt.

Ergebnis 16
Werkzeuge können gesichert werden, indem sie in verschlossenen Behältern aufbewahrt oder selbst abgeschlossen werden.

3.4.5. Automaten

Automaten sind „Arbeitsmittel um im Rahmen einer Aufgabe Material zu bearbeiten“ ([Züllighoven et al. 98]). Automaten arbeiten wie Werkzeuge auf Materialien und Behältern. Der wesentliche Unterschied zu Werkzeugen ist, daß Werkzeuge durch den Benutzer geführt atomare Operationen auf dem Material ausführen, während ein Automat ohne weiteres Eingreifen des Benutzers verschiedene Handlungsschritte ausführt ([Züllighoven et al. 98], Seite 195). Automaten werden von einem Anwender immer über ein Einstellwerkzeug bearbeitet und verrichten ihren Dienst anschließend selbständig ohne weitere Eingriffe durch Anwender (z.B. ein Postversand - Automat).

Wenn der Automat seinen Dienst verrichtet, kann es vorkommen, daß er auf verschlossene Komponenten zugreifen muß. Der Automat kann seinen Dienst nur verrichten, wenn der Anwender, in dessen Arbeitsumgebung sich der Automat befindet, über einen entsprechenden Schlüssel verfügt. Diesen verwendet er, um verschlossene Gegenstände zu öffnen.

Wie in Abschnitt 3.3.1 beschrieben, können sich Automaten aufgrund von Schlüsselschaltern unterschiedlich verhalten. Ein solcher Schlüsselschalter ermöglicht es einem Benutzer, den Automaten mit Hilfe des Einstellwerkzeugs für unterschiedliche Arbeitsweisen zu konfigurieren bzw. ihn überhaupt erst in Betrieb zu nehmen.

Ergebnis 17
Automaten können mit Schlössern versehen werden. Das Schloß verhält sich wie ein Schalter der die Arbeitsweise des Automaten beeinflußt. Automaten können den Schlüssel eines Anwenders verwenden, um verschlossene Gegenstände zu öffnen.

3.4.6. Fachliche Services

Fachliche Services sind ein im WAM-Rahmen neues Konzept, das von M. Otto und N. Schuler in einer Diplomarbeit beschrieben wurde (siehe [Otto, Schuler 00]). Dem *fachlichen Service* liegt die Idee des „Dienstleisters“ zugrunde:

Definition 30: Fachlicher Service
Ein Fachlicher Service faßt Wissen eines Anwendungsbereichs unabhängig von einer Benutzungsschnittstelle zusammen, kapselt es und stellt es als Dienstleistung zur Verfügung. (siehe [Otto, Schuler 00])

Ein Beispiel aus der Anwendungswelt wäre eine Autowerkstatt oder die Zeitansage. Ein fachlicher Service arbeitet direkt oder mittels eines weiteren fachlichen Services auf Materialien. Dabei vereint er das normalerweise in verschiedenen Werkzeugen untergebrachte fachliche Wissen und stellen es zentral zur Verfügung. Fachliche Services ähneln sowohl Werkzeugen als auch Automaten. Im Unterschied zu Werkzeugen werden sie aber nicht unmittelbar durch einen Anwender benutzt, und im Unterschied zu Automaten erbringen sie keine regelmäßigen Dienste und werden nicht von selbst aktiv.

Fachliche Services bieten Funktionalität an, die nicht jeder Anwender nutzen darf. Daher ist es notwendig, auch fachliche Services mit Schlössern zu versehen. Ein abgeschlossener fachlicher Service kann seinen Dienst nicht erbringen. Ähnlich einem Automaten muß er zum Teil auf abgeschlossenen Komponenten (z.B. anderen fachlichen Services) arbeiten. Um diese aufzuschließen, verwendet er die Schlüssel des Anwenders, der letztlich seine Benutzung veranlaßt hat.

> **Ergebnis 18**
> Fachliche Services können mit Schlössern versehen werden und von Anwendern nicht benutzt werden, wenn sie abgeschlossen sind. Soll ein fachlicher Service einen verschlossenen Behälter öffnen, benötigt er hierfür den Schlüssel des Anwenders.

Weiterhin ergeben sich Konsequenzen für Werkzeuge und Automaten: hat ein Anwender keinen Schlüssel für den fachlichen Service, so darf er dessen Funktionalität nicht nutzen. Dies muß ihm durch das Werkzeug direkt (bzw. bei Automaten durch das Einstellwerkzeug) deutlich gemacht werden.

3.5. Veränderungen in der Metapher gegenüber dem Urbild

Verglichen mit Schlüsseln und Schlössern in der Anwendungswelt (dem Urbild) ergaben sich auch Änderungen in der Metapher. Um diese Änderungen hervorzuheben, werden Metapher und Urbild in Tabelle 4 gegenübergestellt:

Urbild	Metapher
Schlösser sind im wesentlichen an Behältern und Türen angebracht. Hinzu kommen Schlüsselschalter (an Automaten) und Schlösser angeketteter Gegenstände.	Schlösser sind an Behältern, Automaten, fachlichen Services sowie der Umgebung angebracht. Zusätzlich können Werkzeuge abgeschlossen werden.
Schlösser und Schlüssel werden erworben. Zertifikate sind nicht notwendig. Einfache Schlüssel können ohne Zertifikat nachgemacht werden.	Es existieren nur Sicherheitsschlüssel. Schlüssel, Schlösser und Zertifikate werden von einem fachlichen Service erzeugt. Zertifikate sind immer notwendig, um Schlüssel und Schlösser zu erzeugen.
Schlüssel können in einer hierarchischen Beziehung zueinander stehen.	keine Änderung gegenüber Urbild
Schlüssel, Schlösser und Zertifikate können weitergegeben werden.	gegebenenfalls Einschränkung, bei Zertifikaten Registrierung des Eingentümerwechsels
Zertifikate, die nicht für einen Eigentümer registriert sind, können weitergegeben werden. Der Besitz eines solchen Zertifikates reicht aus, um Schlüssel nachzumachen.	nicht übernommen
Der Eigentümer eines Zertifikates *kann* registriert werden. Ein Eigentümerwechsel *kann* dem Hersteller mitgeteilt werden.	Der Eigentümer eines Zertifikates *muß* registriert werden. Ein Eigentümerwechsel *muß* dem Hersteller mitgeteilt werden.
Alle Schlösser werden explizit auf-/abgeschlossen.	keine Änderung gegenüber Urbild
Schlüssel und Schlösser können defekt sein.	nicht übernommen
Verliehene Schlüssel und Zertifikate können zurückverlangt werden. Die Rückgabe kann vor Gericht erzwungen werden.	Verliehene Schlüssel und Zertifikate können zurückverlangt werden. Das Herausgeben kann aber nicht erzwungen werden.

Schlüssel und Zertifikate können verlorengehen, beschädigt oder gestohlen werden	nicht übernommen
Einfache Schlüssel können zufällig baugleich sein und passen in dasselbe Schloß.	nicht übernommen

Tabelle 4: Unterschiede zwischen der Schlüssel-Schloß-Metapher und ihrem Urbild

Einige Einschränkungen der Metapher gegenüber dem Urbild sind darin begründet, daß es keinen Sinn macht, Sicherheitsrisiken (z.B. den Verlust eines Schlüssels) zu modellieren. Eine weitere Einschränkung ist, daß nur noch Sicherheitsschlüssel verwendet werden. Der Grund für die Verwendung einfacher Schlüssel in der Anwendungswelt ist, daß Sicherheitsschlüssel teurer in der Herstellung sind. Dieses Argument gilt nicht für Schlüssel innerhalb eines Anwendungssystems. Deshalb werden in der Beschreibung der Metapher lediglich Sicherheitsschlüssel verwendet. Des weiteren ist – wie bereits erwähnt – auch in der Anwendungswelt nicht erwünscht, daß nicht-private Schlüssel (z.B. Büroschlüssel) nachgemacht werden. Innerhalb einer Organisation sollte immer der Eigentümer des Schlüssels gefragt werden, ob er einer Vervielfältigung des Schlüssels zustimmt. Diese Sichtweise wird durch die Verwendung von Zertifikaten konsequent durchgesetzt.

Die Verwendung von Schlössern an Türen und angeketteten Gegenständen wird im Rahmen dieser Arbeit nicht diskutiert, da diese Konzepte kein integraler Bestandteil des WAM-Ansatzes sind. Näheres hierzu findet sich im Zusammenhang mit dem Raumkonzept, wie es in [Roock & Wolf 98] und [Koch 00] beschrieben wird.

Ausgedehnt wird der Umgang mit Schlössern auf Werkzeuge. Begründet wird dies durch eine einfache Handhabbarkeit die sich ergibt, indem Werkzeuge nicht bei jedem Gebrauch aus einem Behälter herausgenommen werden müssen.

Weiterhin ist anzunehmen, daß in einem Anwendungssystem im Vergleich zur Anwendungswelt mehr Gegenstände abgeschlossen werden. In der Anwendungswelt existieren Konventionen und ein Vertrauensverhältnis der Anwender untereinander, wodurch Zugriffskontrollen an vielen Stellen unnötig sind. Die Anwender eines Softwaresystems hingegen nehmen sich gegenseitig nicht wahr und können daher auch nicht darauf vertrauen, daß diese Konventionen eingehalten werden. Einziger Schutz vor ungewollten Zugriffen sind dann vermehrte Zugriffskontrollen. Diese Zugriffskontrollen müssen für die Anwender gut handhabbar sein, damit sie effektiv benutzt werden können. Hierzu sollen die in diesem Abschnitt beschriebenen Veränderungen der Schlüssel-Schloß-Metapher gegenüber ihrem Urbild beitragen.

3.6. Betrachtung anderer Arbeitsplatztypen

Neben dem „Arbeitsplatz für eigenverantwortliche Expertentätigkeit“ werden in [Züllighoven et al. 98] (Seite 94ff) auch andere Leitbilder für die Konstruktion von WAM-Systemen verwendet. Die Werkzeuge an einem solchen Arbeitsplatz dürfen und können vom Anwender meist in vollem Umfang genutzt werden, da dieser ein Experte ist. Das Leitbild hat beim Systementwurf entscheidenden Einfluß auf die Notwendigkeit und Verwendung von Zugriffskontrollen, denn mit dem Leitbild wird ein bestimmter Umgang mit dem System beschrieben. Einige grundlegende Gesichtspunkte, nach denen sich die Arbeitsplatztypen unterscheiden lassen, sind nach [Züllighoven et al. 01]:

- **Umfang der IT-Kenntnisse**: Welche allgemeinen Kenntnisse im Umgang mit fensterbasierten IT-Systemen werden vom Anwender erwartet? Beispiele sind der Umgang mit Drag & Drop oder überlappenden Fenstern. Solche Kenntnisse können heutzutage noch nicht als selbstverständlich vorausgesetzt werden.
- **Vielfalt der Aufgaben:** Ist das Anwendungssystem darauf zugeschnitten vielfältige Aufgaben zu unterstützen?

- **Ausmaß an Flexiblität:** Wie groß ist die Handlungsfreiheit des Anwenders im Umgang mit dem System? Oder umgekehrt: wie gering ist die Ablaufsteuerung?
- **Umfang der Unterstützung:** welche Hilfestellungen müssen Anwender durch das System selbst (z.B. Online-Hilfe), Handbücher, Telefonsupport u.ä. erhalten?

Diese Gesichtspunkte sollen anhand des schon bekannten Leitbildes „Arbeitsplatz für eigenverantwortliche Expertentätigkeit“ erläutert werden (siehe auch Abbildung 13).

Anwender auf solchen Arbeitsplätzen sind hochqualifizierte Experten. Insofern ist davon auszugehen, daß sie IT-Systeme nach einer Schulung gut benutzen können. Ebenfalls hoch sind die Vielfalt der Aufgaben und das Ausmaß an Flexibilität: der Arbeitsablauf wird durch einen ständigen Wechsel verschiedener Tätigkeiten geprägt. Der Anwender entscheidet dabei selbst, was wann zu tun ist. Der Umfang der Unterstützung ist – nach einer mehrwöchigen Einarbeitungsphase – eher als mittelmäßig anzusehen.

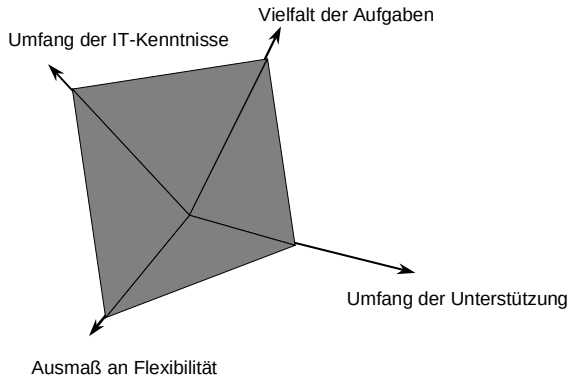


Abbildung 13: Dimensionen des Leitbilds „Arbeitsplatz für eigenverantwortliche Expertentätigkeit“ (aus [Züllighoven et al. 01])

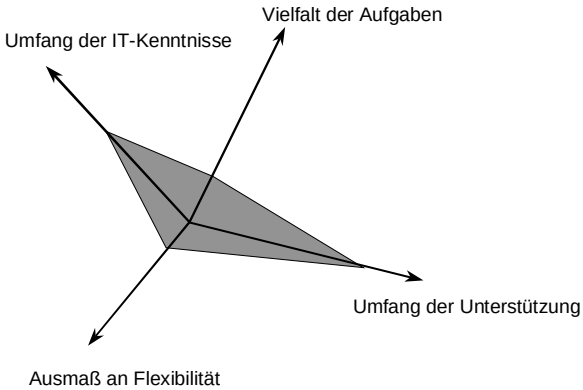


Abbildung 14: Dimensionen des Leitbilds „Funktionsarbeitsplatz“ (aus [Züllighoven et al. 01])

3.6.1. Der „Funktionsarbeitsplatz für eigenverantwortliche Expertentätigkeit“

Funktionsarbeitsplätzen (vgl. Abbildung 14) sind dadurch gekennzeichnet, daß wenige, vorgegebene Aufgaben in häufiger Wiederholung erledigt werden. Zur Unterstützung stehen statt vieler Werkzeuge und Materialien meist nur wenige Spezialwerkzeuge zur Verfügung. Ein Beispiel hierfür ist ein Röntgenanalyse-Arbeitsplatz.

Die einfache Benutzbarkeit steht bei solchen Arbeitsplätzen noch weiter im Vordergrund, da aufgrund der vielen Wiederholungen des Arbeitsablaufes jede Einschränkung vermieden werden muß und nur eine mittlere Erfahrung im Umgang mit IT-Systemen zu erwarten ist. Demnach sollte die Zugriffskontrolle aus Anwendersicht möglichst nicht bemerkbar sein.

Die Sicht der Anwender auf Materialien und Werkzeuge ist bei diesem Leitbild eine andere, als beim bisher verwendeten Leitbild. An Funktionsarbeitsplätzen arbeiten Experten mit spezialisierten Werkzeugen. Die bearbeiteten Materialien sollen anderen Anwendern zügig zur Verfügung gestellt werden. Die Arbeitsergebnisse dürfen aber nicht von jedem Anwender eingesehen werden, daher muß beim Systementwurf darauf geachtet werden, daß die weitergeleiteten Materialien möglichst automatisch und ohne Beteiligung des Anwenders am Funktionsarbeitsplatz in zugriffsgeschützte Behälter gelangen.

Hier ergibt sich ein Problem in Zusammenhang mit der bisher vorgestellten Handhabung von Schlössern: sie können nur explizit aufgeschlossen werden. Für den Funktionsarbeitsplatz macht diese Handhabung keinen Sinn, denn – wie oben erwähnt – soll die Zugriffskontrolle für Anwender möglichst nicht bemerkbar sein. Dies ist sicher nicht der Fall, wenn bei einer Vielzahl von Arbeitsschritten erst Schlösser auf- bzw. abgeschlossen werden müssen. Vielmehr sollte die Maxime gelten, daß die angebotenen Gegenstände ohne weiteres benutzt werden können.

3.6.2. Der „Gruppenarbeitsplatz für eigenverantwortliche, kooperative Aufgabenerledigung“

Ein Gruppenarbeitsplatz (vgl. Abbildung 15), z.B. das Stationszimmer in einer Krankenhausabteilung, wird von vielen Personen mit unterschiedlicher Qualifikation (vom Praktikanten bis zur Chefärztin) benutzt. Kennzeichnend sind eine hohe Aufgabenvielfalt, ein großes Maß an Flexibilität, aber (im Mittel) nur geringe Erfahrungen der Anwender mit IT-Systemen. Die Arbeitsteilung ist hoch und der Informationsaustausch erfolgt häufig über Gegenstände (z.B. Patientenakten).

Eine typische Anwendungssituation ist, daß das Anwendungssystem den ganzen Tag auf wenigen Arbeitsplatzrechnern läuft. Während eines Arbeitstages werden die Arbeitsplatzrechner häufig von unterschiedlichen Anwendern genutzt. Wenn kooperative Arbeit unterstützt werden soll, müssen diese Anwender eine gemeinsame Arbeitsumgebung haben, die im allgemeinen durch einen Desktop dargestellt wird. Die Materialien und Werkzeuge befinden sich dann üblicherweise direkt auf dem Desktop oder in Behältern auf dem Desktop.

Da nicht jeder Anwender alle Werkzeuge und Materialien benutzen darf (z.B. dürfen nur Ärzte Verordnungen verschreiben), sollten solche Materialien und Werkzeuge in verschlossenen Behältern untergebracht werden. Der kooperative/koordinative Aspekt dieses Arbeitsplatzes wird dadurch deutlich, daß alle Anwender über dieselbe Arbeitsumgebung und dieselben Gegenstände verfügen. Nachdem ein Material verändert wurde, wird die Veränderung anderen Anwendern symbolisiert, so daß sie sich selbständig darüber informieren können. Ein Beispiel ist eine elektronische Krankenakte, in der eine neue Verordnung eingetragen wird. An der Krankenakte kann dies durch eine farbige Markierung deutlich gemacht werden.

Die Umgebung präsentiert sich so, wie sie der letzte Anwender hinterlassen hat. Die Zugriffskontrolle spielt eine große Rolle, wenn ein Werkzeug gestartet werden soll, das nicht jeder Anwender starten darf. Die Umgebung verhindert das Starten von Werkzeugen, für die der Anwender keinen Schlüssel besitzt. Aufgrund der sehr unterschiedlichen Qualifikationen der Anwender ist eine möglichst intuitive Bedienbarkeit für das Anwendungssystem wichtig. Diese wird durch die Schlüssel-Schloß-Metapher unterstützt.

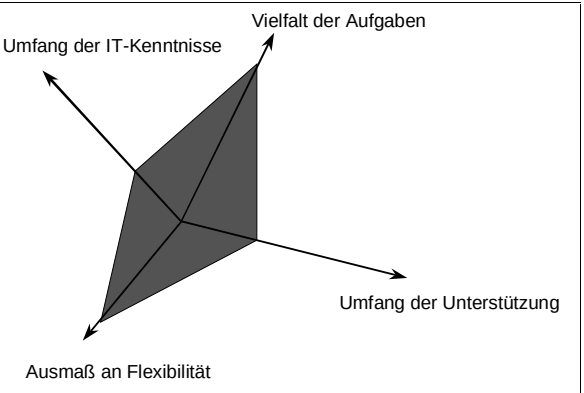


Abbildung 15: Dimensionen des Leitbilds „Gruppenarbeitsplatz“ (aus [Züllighoven et al. 01])

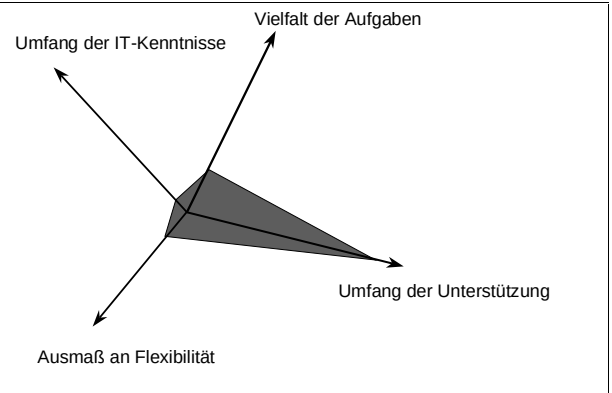


Abbildung 16: Dimensionen des Leitbilds „Selbstbedienungsautomat“ (aus [Züllighoven et al. 01])

3.6.3. Der „Selbstbedienungsautomat“

Ein Selbstbedienungsautomat (vgl. Abbildung 16), z.B. eine Homebanking-Anwendung, wird „nur gelegentlich im Rahmen einer Dienstleistung von einem Kunden in Anspruch genommen“ (siehe [Züllighoven et al. 98]). Der Kunde ist zwar für die Ergebnisse seiner Arbeit voll verantwortlich, aber im Umgang mit dem System meist ungeübt. Der Anbieter der Dienstleistung hat also „eine gewisse Sorgfalts- und Informationspflicht“, die sich zum Beispiel in einer „passenden Benutzerführung“ darstellt. Systeme dieser Art sind meist so gestaltet, daß ein Steuerungsautomat die jeweils möglichen Aktionen vorgibt.

Der Kunde selbst möchte in diesem Fall gar nichts über die konkrete Arbeitsweise von Zugriffskontrollen wissen. Er muß im allgemeinen nur überzeugt sein, daß die vorhandenen Sicherheitsmechanismen ausreichend sind. Das heißt, nur er (sowie Bankangestellte an einem anderen Arbeitsplatz) darf seine Materialien (z.B. sein Konto) bearbeiten und das auch nur, wenn das entsprechende Werkzeug vorher auf seinen Wunsch frei geschaltet wurde. Bezogen auf das Beispiel Homebanking ist es durchaus üblich, daß die verschiedenen Werkzeuge (von der Kontoverwaltung bis zum Optionsschein-Trading-Tool) einzeln freigeschaltet werden.

Nach erfolgter Zugangskontrolle sollte es nicht mehr möglich sein, auf einen Gegenstand zuzugreifen, auf den man keinen Zugriff haben soll. Die Ablaufsteuerung sollte die Möglichkeit solcher Zugriffe gar nicht erst anbieten. Trotzdem müssen die Zugriffe natürlich weiterhin überprüft werden, denn effektiver Zugriffsschutz darf nicht alleine vom Schutz durch das Werkzeug abhängen, mit dem der Anwender arbeitet.

3.6.4. Zusammenhang von Entwurfsmetaphern und Arbeitsplatztypen

Beim Entwurf von Anwendungssystemen kommen – unbeachtet vom konkret unterstützten Arbeitsplatz – alle genannten Entwurfsmetaphern zum Einsatz. Die Wahrnehmung der Metaphern unterscheidet sich allerdings erheblich.

Als Beispiele seien hier ein Berater in einer Bank an einem Expertenarbeitsplatz und ein Bankkunde genannt. Beide arbeiten mit denselben Materialien (z.B. Konto, Überweisung). Der Berater nimmt nach Bedarf Materialien aus Behältern, legt sie in seine Arbeitsumgebung und bearbeitet sie mit Werkzeugen. Um das Auf- und Abschließen der Behälter muß er sich selbst kümmern. Die Sicht des Kunden ist hingegen durch die Werkzeuge geprägt, mit denen er Materialien bearbeiten kann. Das Material als solches wird kaum wahrgenommen. Behälter, Umgebung, Schlösser und Schlüssel sind überhaupt nicht sichtbar.

Die folgende Übersicht zeigt, wie deutlich die verschiedenen Entwurfsmetaphern an den unterschiedlichen Arbeitsplatztypen wahrgenommen werden. Als Bewertungen werden *nicht deutlich* (–), *wenig deutlich* (o) und *sehr deutlich* (+) verwendet.

	Experten-arbeitsplatz	**Funktions-arbeitsplatz**	**Gruppen-arbeitsplatz**	**Selbstbedie-nungsautomat**
Arbeitsplatz	+	o	+	–
Material	+	o	+	–
Werkzeug	+	+	+	–
Behälter	+	o	+	–
Automat	o	–	o	+
fachl. Service	o	–	o	o
Schlüssel/Schloß	+	–	+	–

Tabelle 5: Zusammenhang von Entwurfsmetaphern und Arbeitsplatztypen

Betrachtet man den Zusammenhang von Entwurfsmetaphern und Arbeitsplatztypen vor dem Hintergrund der Relevanz von Zugriffskontrollen für die Entwurfsmetaphern (siehe Abschnitt 3.4), wird die besondere Bedeutung von explizitem Schließen bei Expertenarbeitsplätzen und Gruppenarbeitsplätzen deutlich. Bei diesen Arbeitsplatztypen spielen selbst definierte Zugriffskontrollen eine besonders wichtige Rolle. Bei Gruppenarbeitsplätzen ist hervorzuheben, daß mehrere Anwender eine Arbeitsumgebung zeitlich versetzt nutzen können. Wechseln die Anwender häufig physikalisch ihren Arbeitsplatz, müssen die Arbeitsumgebungen von mehreren Rechnern aus verfügbar sein.

Bei Funktionsarbeitsplätzen und Selbstbedienungsautomaten tritt der direkte Umgang mit Schlüsseln und Schlössern in den Hintergrund. Die Anwender erwarten, daß sie die sichtbaren

Gegenstände auch benutzen können. Zugriffskontrollen müssen daher im Hintergrund stattfinden und werden nicht selbst definiert. Ein Lösungsansatz hierfür wird im folgenden Kapitel vorgestellt.

3.7. Kooperation und Zugriffsschutz

Aus der Betrachtung der verschiedenen Leitbilder wird deutlich, daß es sich bei den WAM-Anwendungssystemen typischerweise um Systeme für vernetzte Einzelarbeitsplätze handelt. Wie auf diesen Einzelarbeitsplätzen mit Zugriffskontrollen mit der Schlüssel-Schloß-Metapher umgegangen werden soll, ist schon beschrieben worden. Durch die Vernetzung kommen aber auch die Aspekte „Verteilung“ und „Kooperation“ zum Tragen: zusätzlich zur lokalen Arbeitsumgebung existiert eine verteilte Umgebung, über die eine Kooperation mit anderen Anwendern möglich ist. Solche Anwendungssysteme können nicht nur als vernetzte Einzelarbeitsplätze, sondern auch als gemeinsam genutzte Arbeitsräume (s.u.) realisiert werden. Die technischen Aspekte von Verteilung werden im Rahmen dieser Arbeit nicht behandelt. Aus fachlicher Sicht bedeutet Verteilung, daß der Anwender auch Gegenstände außerhalb seiner lokalen Arbeitsumgebung verwenden kann. Solche Gegenstände werden im WAM-Ansatz als *Kooperationsmedien* bezeichnet (siehe [Züllighoven et al. 98], Seite 429):

> **Definition 31: Kooperationsmedium**
> Ein Kooperationsmedium ist ein fachlich motivierter Gegenstand, der zur Realisierung von Kooperation in Anwendungssystemen dient. Gemeinsam ist allen Kooperationsmedien, daß mit ihrer Hilfe Materialien oder Informationen ausgetauscht werden können und daß sie selbst vergegenständlicht sind.

Beispiele für Kooperationsmedien sind eine fachliche Registratur und das Postversandsystem (s.u.). Darüber hinaus gibt es auch Gegenstände, an denen sich die Kooperation der Anwender manifestiert. Solche Gegenstände werden *Kooperationsmittel* genannt (vgl. [Züllighoven et al. 98], Seite 429):

> **Definition 32: Kooperationsmittel**
> Ein Kooperationsmittel ist ein fachlich motivierter Gegenstand, der die Kooperation unterstützt. Er vergegenständlicht die Kooperation oder die dabei notwendige Koordination. Beispiele für Kooperationsmittel sind oder Laufzettel.

3.7.1. Implizite und explizite Kooperation

Je nachdem, ob der kooperative Aspekt einer Arbeitssituation deutlich wird oder nicht, spricht man von impliziter oder expliziter Kooperation ([Züllighoven et al. 98], Seite 429ff). Bei impliziter Kooperation läßt sich gegenüber dem Anwender nicht verbergen, daß es konkurrierende Zugriffe auf Materialien gibt. Zwar existieren „mehrere Arbeitsplätze innerhalb einer gemeinsamen Umgebung“, vom einzelnen Arbeitsplatz aus sind diese aber nicht sichtbar. Vor allem koordinieren sich die Beteiligten durch „Konventionen außerhalb des Anwendungssystems“. Bei expliziter Kooperation hingegen arbeiten „mehrere Benutzer kooperativ in einer gemeinsamen Arbeitsumgebung“ ([Züllighoven et al. 98], Seite 437ff). Die Zusammenarbeit wird durch geeignete Kooperationsmittel und –medien unterstützt.

3.7.2. Kooperationstypen

Auch hier gilt, daß sich auf der bisher diskutierten allgemeinen Ebene keine Aussagen über Zugriffskontrollen machen lassen. Daher sollen im folgenden sogenannte „Kooperationstypen“ (siehe [Gryczan et al. 99c] und [Züllighoven et al. 98], Seite 427ff) untersucht werden, die ein konkreteres Modell für die Kooperation anbieten. Kooperationstypen abstrahieren von konkreten Systemen zur Kooperationsunterstützung und beschreiben ihre generellen Eigenschaften.

Implizite Kooperation über eine gemeinsame Registratur

Mit einer Registratur (siehe auch [Havenstein 99], [Gryczan et al. 00]) werden Materialien mehreren Benutzern zur Verfügung gestellt. Diese Benutzer bearbeiten zeitlich versetzt dieselben Materialien. Das Kooperationsmittel stellt hier die Registratur dar. Sie ist dafür verantwortlich, daß immer nur ein Anwender zur Zeit ein Material bearbeitet. Andere Anwender werden bei Bedarf mit Kopien ausgestattet. Das Kooperationsmedium ist entsprechend ein Browser-Werkzeug für das Archiv (vgl. Abbildung 17).

Der Registratur kommt damit die für die Zugriffskontrolle entscheidende Rolle zu. Die einfachste Lösung ist, die Registratur abzuschließen und nur die Anwender mit Schlüsseln auszustatten, die auch Zugriff auf den gesamten Inhalt haben sollen.

Es kann aber auch Situationen geben, in denen Anwender nicht auf alle Materialien innerhalb der Registratur zugreifen dürfen. Dabei wäre es nicht hilfreich, ihnen den Umgang mit der gesamten Registratur zu verbieten. Statt dessen wird hier vorgeschlagen, die Registratur dahingehend zu erweitern, daß sie auch Behälter aufnehmen kann. Behälter sind abschließbar und können schützenswerte Materialien enthalten. Auf die gleiche Weise sollten Informationen, die die Registratur über die Materialien führt (z.B. Inventarverzeichnis, Terminbuch für Wiedervorlagen) behandelt werden.

Die Registratur eignet sich auch dazu, Arbeitsumgebungen zu lagern. Die Arbeitsumgebungen sind dann von verschiedenen Arbeitsplatzrechnern aus zugreifbar. Sofern sie abgeschlossen sind, können sie nur von Anwendern mit einem passenden Schlüssel verwendet werden.

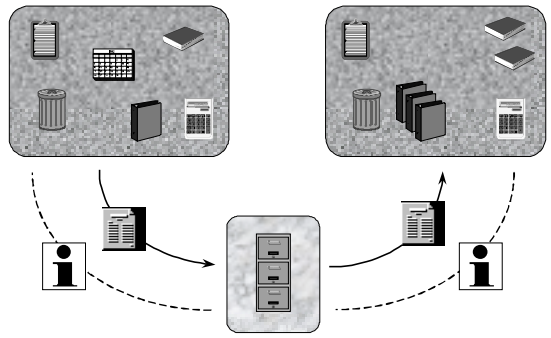


Abbildung 17: Kooperation zwischen zwei Arbeitsplätzen über eine gemeinsame Registratur ([Züllighoven et al. 01])

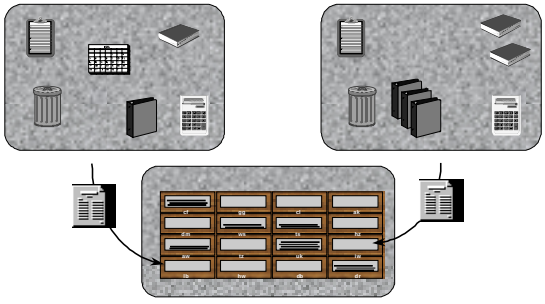


Abbildung 18: Kooperation zwischen zwei Arbeitsplätzen über gemeinsame Postfächer ([Züllighoven et al. 01])

Explizite Kooperation über Postfächer

Postfächer sind ein einfaches Kooperationsmedium, das den Anwendern aus dem Büroalltag geläufig ist. Das Kooperationsprinzip ist einfach: soll ein Anwender ein Material erhalten, wird es in dessen Postfach gelegt (vgl. Abbildung 18). Der Sinn einer Postfachwand ist – neben der reinen Materialübermittlung – daß andere Anwender sehen können, wenn ein Postfach sehr voll ist (z.B. weil ein Kollege krank ist). In diesem Fall ist es gewünscht, daß jeder Anwender auf das Postfach zugreifen kann.

Eine Zugriffskontrolle ist auf dieser Ebene also unnötig. Will man sicherstellen, daß nur ein bestimmter Anwender das Material bekommt, sollte man ein anderes Kooperationsmedium oder einen verschlossenen Behälter verwenden. Es kann aber durchaus sinnvoll sein, die gesamte Postfachwand abzuschließen. In der Anwendungswelt entspräche dies dem Abschließen des Raumes, in dem die Postfächer untergebracht sind. Auf diese Weise könnte man zum Beispiel eine eigene Postfachwand für eine Abteilung eines Unternehmens realisieren.

Explizite Kooperation über direkte Postverbindung

Bei einer direkten Postverbindung (siehe Abbildung 19) wird ein Material direkt von einem Anwender zu einem anderen Anwender bewegt. Da der Absender dem Empfänger bewußt etwas schickt und der Empfänger dies unmittelbar erhält, sind zusätzliche Zugriffskontrollen nicht sinnvoll.

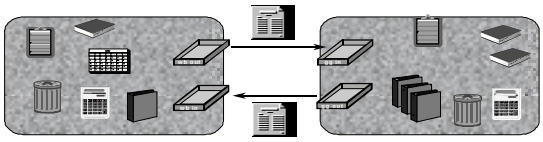


Abbildung 19: Kooperation zwischen Arbeitsplätzen mit direkt verbundenen Postkörben ([Züllighoven et al. 01])

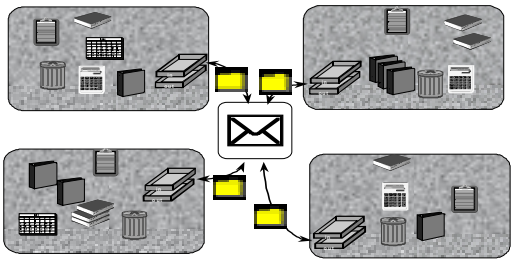


Abbildung 20: Kooperation zwischen mehreren Arbeitsplätzen durch ein Postversandsystem ([Züllighoven et al. 01])

Explizite Kooperation über Postversand

Auch ein Postversandsystem (vgl. [Freund 00], Abbildung 20) stellt ein Kooperationsmedium dar. Der erste Unterschied zur direkten Verbindung ist die notwendige Adressierung und Zwischenspeicherung. Dabei können nicht nur Personen, sondern auch Personengruppen (Rollen) adressiert werden. Im letzteren Fall sorgt ein Verteilungsalgorithmus für die korrekte Zustellung.

Auch bei diesem Kooperationstyp sind zusätzliche Zugriffskontrollen nicht sinnvoll. Das Versenden von Materialien erfordert eine explizite Handlung und es kann davon ausgegangen werden, daß der Anwender weiß, was er tut. Zudem bieten ihm die schon bekannten verschließbaren Behälter genügend Sicherheit an, falls er fürchtet, den falschen Adressaten anzugeben. Allenfalls am *Posteingangskorb* und am *Postausgangskorb*[15] ließen sich hier sinnvoll Schlösser anbringen.

Explizite Koordination durch Laufzettel

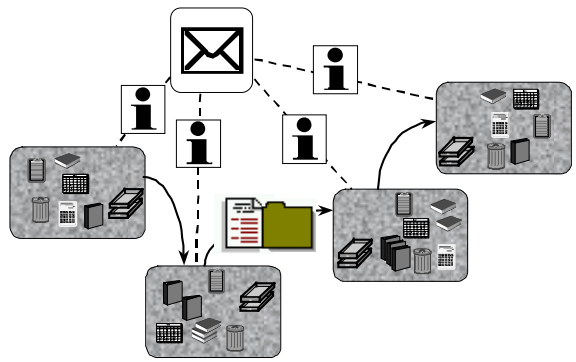


Abbildung 21: Koordination der Kooperation zwischen mehreren Arbeitsplätzen durch Laufzettel ([Züllighoven et al. 01])

Eine einfache und funktionstüchtige Möglichkeit Vorgangsbearbeitungen zu koordinieren, sind Laufzettel ([Züllighoven et al. 98], Seite 449). Auf einem Laufzettel werden die Tätigkeiten, die im Rahmen einer Aufgabe zu erledigen sind, in einer bestimmten Reihenfolge und, mit einem Bearbeiter versehen, aufgeschrieben.

Ein Laufzettel kann dem Postversandsystem übergeben werden, welches ihn automatisch an den Bearbeiter der nächsten zu erledigenden Aufgabe weiterleitet. Während ein Laufzettel abgearbeitet wird, können sich Änderungen in den Aufgaben bzw. bei den Bearbeitern ergeben. In diesem Fall können noch offene Aufgaben verändert und mit einem anderen Bearbeiter versehen werden, abgeschlossene Aufgaben hingegen nicht. Laufzettel sind Kooperationsmittel und werden als Materialien modelliert. Besondere Zugriffskontrollen für Laufzettel sind nicht möglich, da Materialien prinzipiell nicht gesondert

[15] Der Posteingangskorb ist ein Behälter auf dem Desktop eines Anwenders. In ihn gelangt die vom Postversandsystem an den Anwender verschickte Post. Entsprechend wird zu verschickende Post dem Postausgangskorb entnommen.

mit Zugriffsschutz versehen werden sollen (siehe Abschnitt 3.4.3). Aus dem Konzept Laufzettel heraus ergibt sich auch keine Motivation, von dieser Handhabung abzurücken.

3.8. Beurteilung der Schlüssel-Schloß-Metapher

Nachdem die Schlüssel-Schloß-Metapher definiert wurde, soll nun überprüft werden, ob sie als Zugriffskontrollmodell für den WAM-Ansatz geeignet ist. Die Beurteilung soll anhand der in Abschnitt 2.3.3 aufgestellten Kriterien erfolgen.

Verständlichkeit für Anwender

Schlüssel und Schlösser sind Anwendern aus dem täglichen Leben vertraut und in den meisten – wenn nicht sogar allen – Anwendungsbereichen vorzufinden. Mit realen Schlüsseln und Schlössern werden dort fachliche Zugriffskontrollen durchgesetzt. Der grundlegende Umgang mit Schlüsseln und Schlössern ist für Anwender sehr gut verständlich.

Die Beschreibung der Metapher weist aber auch Unterschiede gegenüber dem ursprünglichen Bild von Schlüsseln und Schlössern auf (vgl. Abschnitt 3.5). Diese Veränderungen dienen dem Ziel, den Umgang mit der Schlüssel-Schloß-Metapher einfach und sicher zu gestalten.

> **Ergebnis 19**
> Die Schlüssel-Schloß-Metapher ist – trotz einiger Änderungen gegenüber ihrem Urbild – für Anwender als gut verständlich einzuschätzen.

Flexibilität

Es ist zu erwarten, daß die Schlüssel-Schloß-Metapher für Zugriffskontrollen genauso flexibel einsetzbar ist, wie andere diskretionäre Zugriffskontrollmodelle. Als Anwendungsfelder sind Systeme nach den Leitbildern „Expertenarbeitsplatz“ und „Gruppenarbeitsplatz“ besonders geeignet. Kommt eines der beiden anderen vorgestellten Leitbilder zum Einsatz, kann der explizite Schließmechanismus nicht mehr verwendet werden. Es wurde auf einen Lösungsansatz hingewiesen, der in Kapitel 4 vorgestellt wird.

> **Ergebnis 20**
> Die Schlüssel-Schloß-Metapher ist für den Einsatz in verschiedenen Kontexten geeignet. Vorteile ergeben sich vor allem, wenn die Leitbilder „Expertenarbeitsplatz“ und „Gruppenarbeitsplatz“ verwendet werden.

Betonung der unterstützenden, eigenverantwortlichen Sichtweise

Die Schlüssel-Schloß-Metapher wurde vor dem Hintergrund des Leitbilds „Arbeitsplatz für eigenverantwortliche Expertentätigkeit“ entworfen. Sie ist vor diesem Hintergrund gut geeignet, eigenverantwortliches Arbeiten zu unterstützen. Zugriffskontrollen können für einen privaten bzw. überschaubaren Kontext definiert werden.

> **Ergebnis 21**
> Die Schlüssel-Schloß-Metapher betont den unterstützenden, eigenverantwortlichen Charakter von Anwendungssystemen.

Absicherung von softwaretechnischen Gegenständen

Anwendungsfachliche Gegenstände und Konzepte werden im konkreten Fall als Entwurfsmetaphern interpretiert. In diesem Kapitel wurde beschrieben, wie die Schlüssel-Schloß-Metapher als Zugriffskontroll-Mechanismus für andere Entwurfsmetaphern verwendet werden kann. Daher eignet sich die Schlüssel-Schloß-Metapher gut zum Schutz softwaretechnischer Gegenstände.

> **Ergebnis 22**
> Die Schlüssel-Schloß-Metapher ist zum Schutz softwaretechnischer Gegenstände geeignet.

Damit kann festgehalten werden, das die Schlüssel-Schloß-Metapher die Anforderungen für ein Zugriffskontrollmodell im Rahmen des WAM-Ansatzes erfüllt.

3.9. Zusammenfassung

In diesem Kapitel wurde die Schlüssel-Schloß-Metapher als spezielle Metapher vorgestellt und ihr Verhältnis zu den WAM-Entwurfsmetaphern untersucht. Dabei wurde zunächst vom Leitbild „Arbeitsplatz für eigenverantwortliche Expertentätigkeit“ ausgegangen.

Schlüssel stellen ein fachliches Modell für Rechte dar und Schlösser werden an Gegenständen im Anwendungssystem angebracht (bzw. sind Teil von Gegenständen), die vor unberechtigtem Zugriff geschützt werden sollen. In Bezug auf die WAM-Entwurfsmetaphern ergab sich, daß Schlösser insbesondere bei Behältern und Werkzeugen zum Einsatz kommen können. Aber auch Arbeitsplatz, Automaten und fachliche Services können abgeschlossen werden.

Beim Entwurf eines abschließbaren Gegenstandes muß berücksichtigt werden, wie sich ein abgeschlossenes Schloß auf die Umgangsformen des Gegenstandes auswirken soll. Welche konkrete Auswirkung ein verschlossenes Schloß hat, hängt von dem konkret modellierten Gegenstand ab. Auf jeden Fall sollte dem Anwender deutlich werden, welche Aktionen er auch mit einem verschlossenen Gegenstand ausführen kann.

Die einfache Verwaltung von Schlüsseln und Schlössern wird durch Zertifikate ermöglicht. Nur der Besitzer eines Zertifikates kann Schlösser austauschen, Schlüssel nachmachen oder hierarchische Beziehungen zwischen Schlüsseln herstellen. Der Systemadministrator kann Zugriffskontrollen erzwingen, indem er wichtige Systemkomponenten mit Schlössern versieht und Anwender mit Schlüsseln hierfür, nicht aber mit Zertifikaten ausstattet.

Schlüssel und Schlösser können den Besitzer einfach wechseln und gleichzeitig Eigentum eines Anwenders bleiben. Der Eigentümer eines Zertifikates wird hingegen registriert. Ein Eigentümerwechsel muß explizit gemacht werden und kann vom System verhindert werden, wenn dies aus Sicherheitsgründen in einem konkreten Anwendungssystem nicht erlaubt sein darf.

Bei Experten- und Gruppenarbeitsplätzen können Anwender Zugriffskontrollen für private oder mit anderen Anwendern gemeinsam genutzte Arbeitsumgebungen selbständig definieren und sie so ihrem individuellen Schutzbedürfnis anpassen. Für Funktionsarbeitsplätze und Selbstbedienungsautomaten sind selbst definierte Zugriffskontrollen nicht vorgesehen und bei ihnen kann das Benutzungsmodell „explizites Schließen“ nicht angewendet werden. Einen Ansatz zur Lösung dieses Problems wird im folgenden Kapitel beschrieben.

Aus der Betrachtung der Auswirkung von Zugriffskontrollen auf Kooperationstypen ergab sich, daß die Registratur dahingehend erweitert werden sollte, daß sie Behälter aufnehmen kann. Da Behälter abschließbar sind, würde man ein Persistenzmedium mit fachlichen Zugriffskontrollen erhalten.

Ein wesentlicher Bestandteil der Zugriffskontrolle ist ein fachliches Modell eines Benutzers, welches ebenfalls in diesem Kapitel vorgestellt wurde. Er ist Besitzer von Zertifikaten, Schlüsseln und Schlössern. Wenn Benutzer ihre Schlüssel untereinander austauschen können, ist zumindest teilweise für eine Vertreterregelung gesorgt. In diesem Fall ist jedoch nicht mehr eindeutig nachvollziehbar, welcher Anwender eine Aktion im System veranlaßt hat. Dann ist eine Protokollierung von Zugriffen besonders wichtig, da Anwender sonst nicht mehr für bestimmte Aktionen verantwortlich gemacht werden können (vgl. Abschnitt 4.5).

Es wird bewußt in Kauf genommen, daß Anwender Einfluß auf Zugriffskontrollen im System nehmen können. Handelt es sich um besonders sicherheitskritische Systeme oder werden andere Leitbilder zur Gestaltung des Arbeitsplatzes verwendet, kann dieser Einfluß verringert oder abgeschafft werden. In diesem Fall können Anwender keine Schlüssel weitergeben und auch keine Zertifikate herstellen oder weitergeben.

Kapitel 4 - Umsetzung in den Entwurf

Softwareentwicklung nach dem WAM-Ansatz ist durch die Verwendung von Leitbildern und Entwurfsmetaphern geprägt. Diese helfen Anwendern und Anwendungsentwicklern, eine allgemeine Sichtweise sowie eine Diskussionsgrundlage über den Anwendungsbereich zu schaffen (siehe [Züllighoven et al. 98], Seite 163). Die Schlüssel-Schloß-Metapher ist im vorangegangenen Kapitel so beschrieben worden, daß sie für Anwender und Anwendungsentwickler auf dieser Ebene dienlich sein kann.

In diesem Kapitel soll untersucht werden, wie die Schlüssel-Schloß-Metapher in einen softwaretechnischen Entwurf umgesetzt werden kann. Technischer Hintergrund dieser Betrachtungen ist dabei das JWAM-Rahmenwerk, welches die Entwicklung von WAM-Systemen unterstützt (siehe Abschnitt 1.2).

Darüber hinaus wird die Schlüssel-Schloß-Metapher als fachliches Zugriffskontrollmodell gegenüber dem technischen Sicherheitsmodell von Java abgegrenzt. Aus diesen Betrachtungen ergeben sich Ansätze zur Beantwortung der Frage, wie ein technisches Sicherheitsmodell fachliche Zugriffskontrollen unterstützen kann (siehe Offene Frage 1).

4.1. Vorstellung des Rahmenwerks

Dieser Abschnitt soll einen Überblick über die Umsetzung einiger grundlegender Konzepte des WAM-Ansatzes in das JWAM-Rahmenwerk geben. Im Rahmen dieser Arbeit ist es nicht möglich, einen vollständigen Einblick in alle Teile des Rahmenwerks zu nehmen. Einen Überblick verschafft zum Beispiel die in der Zeitschrift OBJEKTspektrum erschienene Artikelserie „Frameworkbasierte Anwendungsentwicklung“ in sechs Teilen[16]. Detaillierte Informationen sowie das Rahmenwerk selbst sind unter [JWAM 00] zu finden.

4.1.1. Hintergrund

Wenn ein neues Anwendungssystem eingeführt werden soll, stellt sich die Frage, ob es sich um Standardsoftware oder eine individuell zugeschnittene Software handeln soll. Für den Fall, daß ein neues Anwendungssystem entwickelt wird, ergibt sich die Forderung nach „effizienten und kostensparenden Entwicklungsprozessen“ (siehe [Gryczan et al. 99a]). Deshalb werden nach Möglichkeit Lösungen wiederverwendet, die bereits für andere Anwendungsbereiche entwickelt wurden. Sofern es sich dabei um objektorientierte Anwendungssysteme handelt, werden diese Lösungen in Form eines Rahmenwerkes (engl. *Framework*) bereitgestellt. Dabei handelt es sich meist um sogenannte *Application-Frameworks*, die auf einen bestimmten Anwendungsbereich (z.B. das Bankgewerbe) zugeschnitten sind.

> **Definition 33: Rahmenwerk**
> Ein Rahmenwerk stellt eine softwaretechnische Lösung für eine Reihe verwandter Probleme zur Verfügung. Statisch besteht es aus einer Menge von Klassen, die, bezogen auf das zu lösende Problem, das Zusammenspiel (Kontrollfluß) einer Gruppe von Objekten zur Laufzeit eines Softwaresystems festlegen. Im Unterschied zu Entwurfsmustern sind Rahmenwerke implementierte Einheiten. Sie ermöglichen somit die Wiederverwendung von Analysen, Entwurfsentscheidungen, Architekturen und Implementierungen. Ein Rahmenwerk definiert dabei auch die Stellen, an denen seine Funktionalität erweitert und angepaßt werden kann. (siehe [Bäumer 98], Seite 93)

Technisch geschieht die Erweiterung und Anpassung entweder durch Spezialisierung (*White-Box* Rahmenwerk) oder durch Verwendung und Parametrisierung (*Black-Box* Rahmenwerk)

[16] Literaturverweise: [Gryczan et al. 99a], [Gryczan et al. 99b], [Bleek et al. 99a], [Bleek et al. 99b], [Gryczan et al. 00a], [Gryczan et al. 00b]

von Klassen des Rahmenwerks. Fachlich wird das Anwendungssystem der Idee des Rahmenwerks entsprechend gegliedert und modelliert.

Die Idee hinter dem JWAM-Rahmenwerk ist der WAM-Ansatz. Das Rahmenwerk vereint Gemeinsamkeiten „generisch jenseits eines konkreten Anwendungsbereichs“ ([Gryczan et al. 99a]) und ist damit kein Application-Framework. Die Modellierung des Anwendungsbereichs wird durch die Konkretisierung von Entwurfsmetaphern in Form von Klassen unterstützt. So gibt es zum Beispiel Sub-Rahmenwerke zur Werkzeug- und Behälterkonstruktion.

Das Rahmenwerk wird seit 1997 in Studien- und Diplomarbeiten am Arbeitsbereich Softwaretechnik der Universität Hamburg von Studenten und wissenschaftlichen Mitarbeitern entwickelt. Zum Zeitpunkt der Verfassung dieser Arbeit liegt es in der Version 1.5 vor, umfaßt ca. 700 Klassen und Schnittstellen und basiert auf dem JDK (Java Development Kit) 1.2.2.

4.1.2. Die Architektur des JWAM-Rahmenwerkes

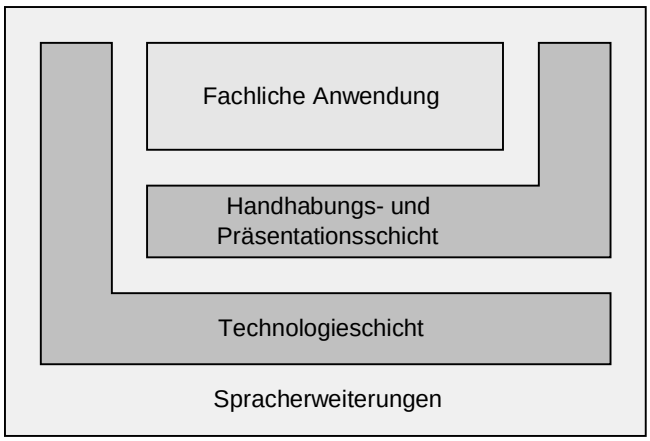


Abbildung 22: Schichtenarchitektur des JWAM-Rahmenwerks

Dem JWAM-Rahmenwerk liegt eine Schichtenarchitektur zugrunde, die ursprünglich in [Bäumer 98] beschrieben wurde (vgl. Abbildung 22). In Abweichung von dem dort Vorgestellten wird auf die Systembasisschicht verzichtet, da die Programmiersprache Java die notwendigen technischen Abstraktionen enthält (z.B. von einer konkreten Datenbank durch JDBC).

Eine Schicht beinhaltet softwaretechnische Komponenten. Jede Schicht bietet einer höher liegenden Schicht eine Leistung an. Dabei verwendet sie ausschließlich die unter ihr liegenden Schichten und kennt die höher liegenden Schichten nicht. Die Schichten bilden so „Abstraktionsstufen“, wobei eine höher liegende Schicht von der tiefer liegenden Schicht abstrahiert (siehe [Bäumer 98], Seite 102). So soll zum Beispiel die fachliche Anwendung möglichst wenig von den zugrundeliegenden Technologien (z.B. Fenstersystem, Datenbanksystem) beeinflußt werden, während die Technologieschicht keine Annahmen über einen konkreten Einsatzkontext macht.

Spracherweiterungen
Obwohl zum Sprachumfang von Java bereits umfangreiche Rahmenwerke gehören (z.B. das Fensterklassen-Rahmenwerk Swing), sind einige aus Sicht der JWAM-Architekten wichtige Konzepte nicht enthalten. Hierzu gehören vor allem das Vertragsmodell (siehe Abschnitt 4.1.9) sowie das Fachwert-Konzept (siehe Abschnitt 4.1.1).

Technologieschicht
Diese Schicht enthält „Modelle der verwendeten Technik“, die aber von den konkret verwendeten Techniken in der Systembasisschicht abstrahieren. Wird z.B. Persistenz benötigt, ist die Anbindung an eine relationale Datenbank in der Systembasisschicht realisiert. Die Technologieschicht wiederum stellt nun nur noch einen Persistenzmechanismus zur Verfügung, der nicht mehr erkennen läßt, welche Art von Speichermedium (z.B. Datei, relationale oder objektorientiert Datenbank) verwendet wird.

Handhabungs- und Präsentationsschicht
Hier sind die wiederverwendbaren Anteile interaktiver Anwendungssoftware untergebracht. Dies sind z.B. Klassen, die die Werkzeugkonstruktion im allgemeinen unterstützen und solche, die häufig verwendete WAM-Entwurfsmuster implementieren ([Züllighoven et al. 98], Seite 200ff). Ein Beispiel ist die Trennung von Funktion und Interaktion bei der Werkzeug-

konstruktion. Hier angesiedelte Sub-Rahmenwerke können z.B. den Umgang mit Strukturierungselementen (Behälter, Mappen, Stapel) zur Organisation des Arbeitsplatzes unterstützen.

4.1.3. Die Package-Struktur

Wie bereits erwähnt, ist das JWAM-Rahmenwerk in Sub-Rahmenwerke untergliedert. Die oberste Ebene ist, wie in Deutschland üblich, die de.-Ebene. Darunter finden sich die Packages jwam, jwamalpha, jwamdev, jwamexample und jwamx, deren Inhalt in der folgenden Tabelle dargestellt wird:

de	jwam	Der Rahmenwerkskern umfaßt z.Zt. ca. 120 Klassen und Schnittstellen. Innerhalb dieses Package findet sich die Schichtenarchitektur aus Abschnitt 4.1.2 wieder. Hier werden grundsätzliche Konzepte, wie z.B. das `Thing`-Konzept (siehe Abschnitt 4.1.5) oder die Umgebung, zur Verfügung gestellt. Klassen aus dem Rahmenwerkskern verwenden keine Klassen aus anderen JWAM-Sub-Rahmenwerken.
	jwamx	Ähnlich wie das jwam-Package ist dieses Package entsprechend der Schichtenarchitektur gegliedert. Hier finden sich Ergänzungen des Kerns, die bei der Anwendungsentwicklung verwendet werden können, aber nicht müssen (z.B. der JWAM-Desktop).
	jwamalpha	Hier befinden sich Sub-Packages, die sich noch in der Entwicklung befinden. Sie sollen zum Kern oder zu den Erweiterungen hinzugefügt werden, sobald sie ausgereift sind.
	jwamexample	Hier sind Beispiele angesiedelt, die die Verwendung der anderen Packages demonstrieren.
	jwamdev	Klassen zur Unterstützung des Entwicklungsprozesses.

Tabelle 6: Die Package-Struktur des JWAM-Rahmenwerkes

4.1.4. Fachwerte

Bei Fachwerten (engl. *domain values*) handelt es sich um ein Konzept, mit dem in objektorientierten Sprachen (z.B. Java und C++) abstrakte Größen zum Abzählen, Ordnen oder zur Identifikation dargestellt werden können. Ein Fachwert ist ein benutzerdefinierter Wert mit definierter Wertemenge und festgelegten Operationen, welcher Werte eines Anwendungsbereiches repräsentiert. Da in einer objektorientierten Sprache keine anderen Ausdrucksmittel zur Verfügung stehen, werden Fachwerte als Klassen definiert. Fachwerte bieten mehr an Funktionalität und Sicherheit in der Benutzung als Basisdatentypen (z.B. Integer). Die erzeugten Exemplare besitzen immer Wertsemantik[17]. Die Darstellungen beruhen auf [Bleek et al. 99b], [Müller 99] und [Züllighoven et al. 98].

Entscheidende Unterschiede zwischen Wert und Objekt sind in Abbildung 23 dargestellt (zum Begriff referentielle Transparenz siehe [Züllighoven et al. 98], Seite 60). Konzeptionell sind unveränderliche Objekte Fachwerten ähnlich. Ein unveränderliches Objekt wird bei seiner Erzeugung auf Basis von anderen Werten (Basisdatentypen oder andere Fachwerte) initialisiert. Eine genaue Abbildung kann mit den bisher verfügbaren objektorientierten Sprachen nicht gelingen. So hat zum Beispiel jedes Objekt einen Lebenszyklus und eine Identität und damit keinen Wertcharakter.

Dem Anwendungsentwickler bietet das JWAM-Rahmenwerk Unterstützung zur Implementierung eigener Fachwerte. Die Fachwerte werden als Objekte modelliert, die sich nach Wertsemantik verhalten. Dem Anwendungsentwickler werden eine Schnittstelle und eine Teilimplementierung angeboten, mit deren Hilfe er eigene Fachwerte realisieren kann. Die Schnitt-

[17] Wertsemantik bedeutet, daß Variablen einen Wert zugewiesen bekommen und auf Gleichheit geprüft werden können. Bei Referenzsemantik hingegen werden Variablen Objekte zugewiesen und die Objekte auf Identität und Gleichheit geprüft.

stelle beinhaltet die Definition einer Fabrik (siehe [Gamma et al. 96]). Konkrete Implementierungen von Fachwerten besitzen keinen öffentlichen Konstruktor, da die Erzeugung von Fachwerten nicht willkürlich erfolgen darf.

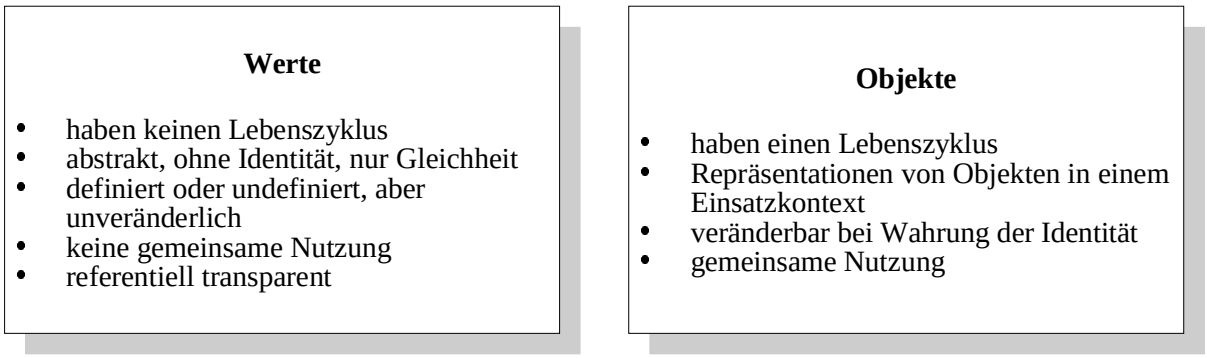


Abbildung 23: Vergleich von Werten und Objekten (nach [Müller 99])

Fachwerte können sehr komplex sein und auch auf modernen, gut ausgestatteten Arbeitsplatzrechnern den Hauptspeicher schnell belegen, wenn nur entsprechend viele Werte erzeugt werden. Daher werden von der Fabrik auf Anfrage neue Fachwerte (bzw. Referenzen auf bereits existierende Fachwerte) zurückgegeben. Ein weiterer Vorteil hiervon ist, daß die Erzeugung von Fachwerten durch die Fabrik einheitlich geregelt wird. Auf die technische Realisierung des Fachwertkonzeptes soll an dieser Stelle nicht näher eingegangen werden.

Ein im Rahmenwerk häufig verwendeter Fachwert ist `dvIdentificator`[18]. Ein `dvIdentificator` ist ein systemweit eindeutiger Wert, der sich niemals ändert, egal wann und in welchem Kontext er betrachtet wird. Ein `dvIdentificator` wird oft zur fachlichen Identifizierung von Gegenständen verwendet.

Ein anderes häufiges Anwendungsgebiet für Fachwerte sind *fachliche Verweise* zwischen Gegenständen:

> **Definition 34: fachlicher Verweis**
> Ein fachlicher Verweis stellt eine fachliche Verbindung zwischen zwei Objekten her. Eine Verbindung durch technische Referenzen oder Zeigervariablen existiert nicht.

Ein Beispiel für einen fachlichen Verweis ist eine Kundennummer. Im Unterschied zu einer technischen Referenz, kann man über den fachlichen Verweis nicht direkt auf das andere Objekt zugreifen. Um an das fachlich referenzierte Objekt zu gelangen, muß man sich mit dem fachlichen Verweis an ein weiteres Objekt wenden, das den fachlichen Verweis auflösen kann. Auf diese Weise wird auch das Problem der zyklischen Objektreferenzen vermieden. Ein Beispiel hierfür ist die Registratur (vgl. Abschnitt 3.7.2). In der Registratur werden Materialien gelagert. Diese können von außerhalb der Registratur nicht als Objektreferenz, sondern nur über den fachlichen Verweis referenziert werden.

4.1.5. Gegenstände im softwaretechnischen Modell

Wie bereits im vorangegangenen Kapitel beschrieben wurde, werden anwendungsfachliche Gegenstände und Konzepte im softwaretechnischen Modell als Entwurfsmetaphern interpretiert und entsprechend modelliert. In Anlehnung an [Roock & Wolf 98] wird als Oberbegriff der Begriff *Gegenstand* verwendet. Abbildung 24 zeigt die Begriffshierarchie und die Beziehungen der Begriffe Werkzeug, Material, Automat, Behälter und fachlicher Service zueinander. Dabei handelt es sich nicht um ein Klassendiagramm.

[18] Im JWAM-Rahmenwerk werden per Konvention grundsätzlich englisch-sprachige Bezeichner verwendet. Die Namen von Fachwerten sind darüber hinaus mit dem Präfix „dv“ als Abkürzung für *domain value* versehen.

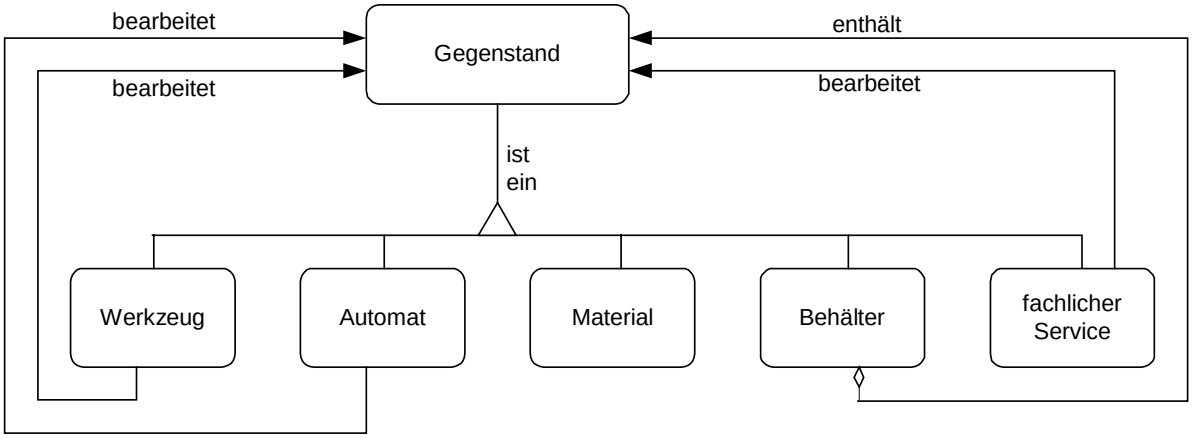


Abbildung 24: Begriffshierarchie der Entwurfskonzepte

Als gemeinsame Schnittstell für Gegenstände wurde im Rahmenwerk die Schnittstelle `Thing` eingeführt (siehe [JWAM 00]). Die Standard-Implementation von `Thing` (`ThingImpl`) schafft technische Voraussetzungen, die die Konstruktion von Gegenständen unterstützen. Ein `Thing` kann eine Beschreibung von sich selbst durch den Fachwert `dvThingDescription` herausgeben. Dieser enthält zur Repräsentation einen Namen und ein Icon sowie zur Identifikation einen Fachwert `dvIdentificator`. Zwei `ThingImpl`-Exemplare sind gleich, wenn ihre `dvIdentificator` Fachwerte gleich sind.

Auf Ebene der `Thing`-Schnittstelle bietet das Rahmenwerk reichhaltige Unterstützung zum technischen Umgang mit fachlichen Gegenständen. Wird z.B. ein Material auf den Desktop bewegt, kann der Desktop auf der `Thing`-Schnittstelle mit ihm arbeiten und es darstellen.

4.1.6. Konstruktion von Behältern

In bisherigen Rahmenwerken war die Unterstützung der Konstruktion technischer Behälter durch ein Sub-Rahmenwerk oft ein zentrales Thema. Für die Vorgänger des JWAM-Rahmenwerkes wurde dies z.B. durch die ConLib (siehe [Traub 95] und [Bohlmann 98]) realisiert. Mit der Einführung einer einheitlichen Behälter-Hierarchie im JDK 1.2 ist eine technische Unterstützung einfacher Behälterklassen (z.B. `Set, List`) gegeben und muß nicht mehr durch das Rahmenwerk angeboten werden.

Auf Ebene der Handling-Schicht steht mit `Thing` eine gemeinsame Schnittstelle für anwendungsfachlicher Gegenstände und Konzepte zur Verfügung. Im Package de.jwam.handling.containerconstruction findet man daher mit der Klasse `ContainerImpl` Unterstützung zur Konstruktion fachlicher Behälter auf der Ebene von `Thing`.

Der wesentliche Unterschied zwischen technischen und fachlichen Behältern ist, daß an der Behälterschnittstelle nur fachliche Operationen zur Manipulation des Behälters vorhanden sind. Diese sind mit der Schnittstelle eines technischen Behälters „nicht ohne weiteres konform“ ([Züllighoven et al. 98], Seite 305). Der intern verwendete technische Behälter wird komplett gekapselt, so daß von außen nicht erkennbar ist, welche Datenstruktur zur Speicherung der enthaltenen Elemente verwendet wird. Auch ein generischer Zugriff auf ein bestimmtes Element ist nicht möglich. Ein erheblicher Vorteil bei der Verwendung fachlicher Behälter ist, daß die interne Datenstruktur geändert werden kann, ohne daß eine andere Klasse, die den Behälter benutzt, überarbeitet werden muß.

Eine häufig benötigte fachliche Operation ist die Abfrage eines Inhaltsverzeichnisses für den Behälter. `ContainerImpl` kann dieses Inhaltsverzeichnis generisch auf Grundlage der `dvThingDescriptions` (siehe Abschnitt 4.1.5) erstellen und als Fachwert herausgeben. Damit ist es besonders einfach, Werkzeuge für fachliche Behälter zu bauen. Diese können dessen Inhalt generisch über die `dvThingDescriptions` ermitteln und anzeigen.

4.1.7. Konstruktion von Werkzeugen

Werkzeuge haben im WAM-Kontext eine Funktion sowie eine Interaktion. Komplexere Werkzeuge werden in eine Funktionskomponente (FK, engl. *functional part*) und eine Interaktionskomponente (IAK, engl. *interaction part*) aufgeteilt (vgl. Abbildung 25). Bei einfachen Werkzeugen werden Funktion und Interaktion in einer Klasse modelliert, die die Rolle von FK und IAK zugleich spielt. Die FK arbeitet auf dem Material und kapselt die fachliche Funktionalität des Werkzeugs (z.B. das Ändern einer Kontonummer). Die IAK registriert Benutzerinteraktionen, ruft die zugehörigen fachlichen Funktionen an der FK auf und präsentiert dem Benutzer den Materialzustand. Wie diese Präsentation geschieht (z.B. auf der Eingabekonsole oder einer Fensterumgebung) ist nur der IAK bekannt.

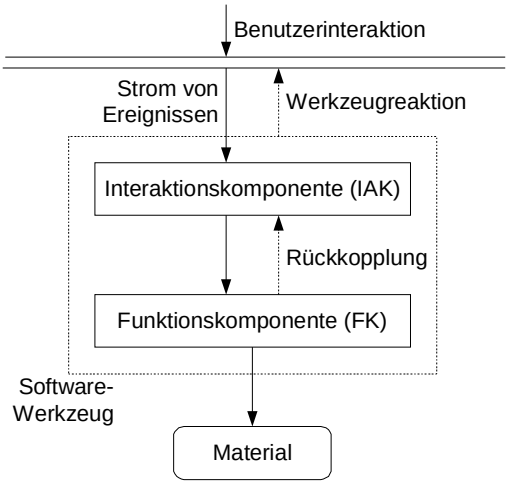


Abbildung 25: Schematische Sicht auf ein komplexes Softwarewerkzeug (nach [Züllighoven 98], Seite 247)

Die FK ist von der konkreten GUI und die IAK vom konkreten Material unabhängig. Die GUI besteht in einem Fenstersystem aus einer Reihe von Bedien- und Anzeigeelementen. Wird z.B. ein Knopf betätigt, ruft die IAK in der FK die zum Knopfdruck gehörende fachliche Funktion (z.B. „Kontonummer ändern“) auf. Die FK bearbeitet das Material entsprechend. Ändert sich das Material während der Bearbeitung, zum Beispiel weil es zwischenzeitlich von einem anderen Werkzeug manipuliert wurde, teilt die FK dies der IAK mit. Die IAK sorgt dann für die Repräsentation in der GUI. Die IAK wird mit der FK durch das Entwurfsmuster „Rückkopplung zwischen Funktion und Interaktion“ verbunden (siehe [Züllighoven et al. 98], Seite 246ff).

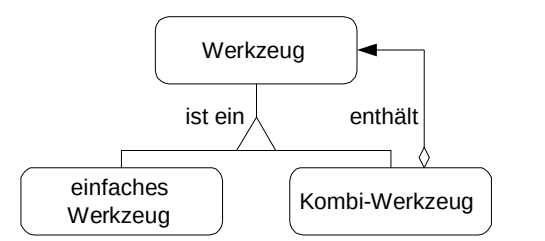


Abbildung 26: Werkzeuge als Sub- und Kombi-Werkzeuge (nach [Züllighoven et al. 98], Seite 280)

Ziel der Werkzeug-Konstruktion ist es, ein passendes Werkzeug für eine klare Aufgabenstellung zu entwerfen. In der Praxis existieren häufig aber komplexe Werkzeuge, mit denen sich mehrere Aufgaben aus einem Arbeitszusammenhang erledigen lassen. Durch Werkzeugkomposition ist es daher möglich, Werkzeuge in andere Werkzeuge einzubetten (siehe [Züllighoven et al. 98], Seite 278ff). Komplexe Werkzeuge können dann wie „Bausteine“ aus einfacheren Werkzeugen zusammengesetzt werden. Zur Unterscheidung werden in [Züllighoven et al. 98] die Begriffe *einfaches Werkzeug*, *Kombiwerkzeug*, *Subwerkzeug* und *Kontextwerkzeug* verwendet (siehe auch Abbildung 26):

- *einfache Werkzeuge* besitzen eine elementare Funktionalität und keine Subwerkzeuge
- *Kombiwerkzeuge* können verschiedene Dienstleistungen erledigen und haben eingebettete *Subwerkzeuge*
- *Kontextwerkzeuge* bilden den technischen und fachlichen Abschluß und realisieren damit Kombiwerkzeuge.
- *Subwerkzeuge* sind in andere Werkzeuge eingebettete Werkzeuge[19]

[19] *Eingebettet* bedeutet dabei funktionell eingebettet. Subwerkzeuge können sowohl eigenständig als auch als Bestandteil des Kontextwerkzeugs dargestellt werden.

Gestartet wird ein Werkzeug durch den Kontext, in den es eingebettet ist. Bei eingebetteten (also Sub-) Werkzeugen ist dies das Kontextwerkzeug, bei einfachen bzw. Kontextwerkzeugen die allgemeine Umgebung.

Im JWAM-Rahmenwerk findet sich Unterstützung zur Werkzeugkonstruktion im Package de.jwam.handling.toolconstruction.

4.1.8. Die Modellierung der Umgebung

Auch die Umgebung, wie sie in Abschnitt 3.4.1 als Entwurfsmetapher beschrieben wird, ist im Rahmenwerk modelliert. Die Unterscheidung der Konzepte Umgebung/Arbeitsplatz/ Desktop wurde beibehalten und wird zur Veranschaulichung in Abbildung 27 dargestellt. Umgebung und Arbeitsplatz werden durch die Klassen `Environment` bzw. `Workspace` modelliert und sind Bestandteile des Package de.jwam.handling.environment. Der Desktop hingegen ist eine den Rahmenwerkskern erweiternde Komponente und Bestandteil des de.jwamx.handling.desktop Package.

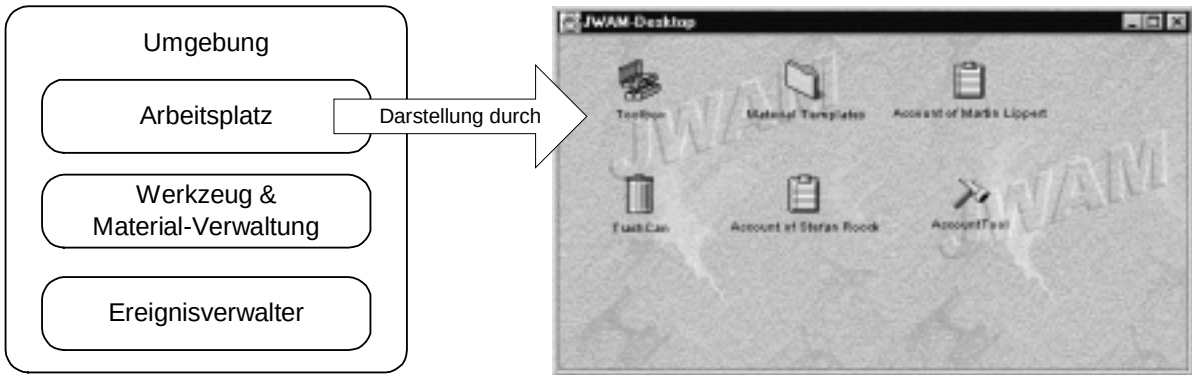


Abbildung 27: Darstellung des Arbeitsplatzes durch die Umgebung

Die Umgebung ist darüber hinaus für die Verwaltung von Werkzeugen und Materialien sowie die Behandlung bestimmter Ereignisse verantwortlich. Unter anderem ist sie in der Lage, auf Anfrage Werkzeuge zu starten und zu schließen, sowie Materialien zu speichern. Des weiteren benachrichtigt sie Werkzeuge, falls ein von ihnen dargestelltes Material von einem anderen Werkzeug verändert wurde.

4.1.9. Das Vertragsmodell

Das Vertragsmodell nach [Meyer 97] beschreibt eine Möglichkeit, die Verläßlichkeit großer Softwaresysteme zu erhöhen. Als Grundidee sollen bei dem Entwurf von Klassen und ihrer Interaktionen nicht nur die Schnittstellen berücksichtigt werden. Die interagierenden Klassen werden als Vertragspartner mit Verpflichtungen und Rechten gesehen:

> **Definition 35: Design by Contract**
> Viewing the relationship between a class and its clients as a formal agreement, expressing each party's rights and obligations. Only through such a precise definition of every module's claims and responsibilities can we hope to attain a significant degree of trust in large software systems. (aus [Meyer 97], Seite 331)

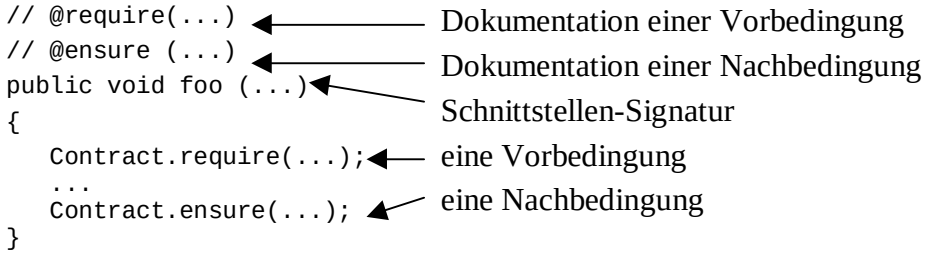
Ein solcher Vertrag kann auf Ebene von Operationen formuliert werden. Nach [Hoare 69] wird die Form {P} A {Q} verwendet, womit zum Ausdruck gebracht wird, daß jeder Aufruf der Operation A, die in einem Zustand P beginnt, in einem Zustand Q endet. Dabei ist P die Vorbedingung (engl. *precondition*) und Q die Nachbedingung (engl. *postcondition*) der Operation A. Entscheidend ist nun, wie P und Q formuliert sind. Aus P sollte hervorgehen, was erfüllt sein muß, damit die Operation A eine Dienstleistung erbringen kann. Diese Dienstleistung ist in Q formuliert.

Vor- und Nachbedingungen stellen eine Möglichkeit dar, die Funktionalität einer Operation über die Signatur hinausgehend zu beschreiben. Dies ist in großen Softwaresystemen sehr

wichtig, denn Klassenmethoden werden hier häufig überarbeitet oder (gerade in einem Rahmenwerk) in einer Unterklasse reimplementiert. Ohne Verwendung von Vor- und Nachbedingungen läßt sich nicht sicherstellen, daß den Klienten der Methoden immer noch dieselbe Funktionalität angeboten wird.

Aufgrund der großen Bedeutung von Vor- und Nachbedingungen und ihrem Fehlen als sprachliches Ausdrucksmittel in Java, ist eine Implementation des Vertragsmodells ein Bestandteil des JWAM-Rahmenwerkes. Nahezu jeder Methodenaufruf ist durch Vorbedingungen abgesichert und nahezu jede Methode stellt Nachbedingungen sicher. Sondierende Funktionen, die den Zustand eines Objektes nicht verändern, sind nicht mit Vor- und Nachbedingungen versehen.

Die Implementation des Vertragsmodells kann im Package de.jwam.lang.contract gefunden werden. Die Operationen `Contract.require(...)` und `Contract.ensure(...)` prüfen einen boolschen Ausdruck und werfen eine Exception, wenn dieser als Ergebnis `false` hat. Die Verwendung und Dokumentation im Quellcode wird in Code 1 demonstriert:



Code 1: Verwendung des JWAM-Vertragsmodells

Bei Verletzung einer Vor- oder Nachbedingung wird eine Exception ausgelöst. Eine Exception unterbricht den normalen Programmverlauf, nicht jedoch das Anwendungsprogramm selbst. Im weiteren Verlauf der Arbeit finden sich Vertragsbedingungen in Codebeispielen, soweit diese zum Verständnis des Codes relevant sind. Nicht relevant ist z.B. die Überprüfung von Parametern und Rückgabewerten auf `null`, obwohl sie natürlich innerhalb des Rahmenwerks durchgeführt werden.

4.2. Umsetzung des fachlichen Modells des Benutzers

Dieser Abschnitt beschreibt, wie sich das Modell des Benutzers in das JWAM-Rahmenwerk sowohl fachlich als auch technisch einbetten läßt.

4.2.1. Ein technischer Lösungsansatz im Rahmenwerk

Wie bereits in Abschnitt 3.1 erwähnt wurde, gab es bisher im Rahmen des WAM-Ansatzes kein fachliches Benutzermodell. Für die Realisierung einiger technischer Komponenten wurde allerdings ein technisches Benutzermodell benötigt. Dieses bestand aus einem einfachen Fachwert `dvUser`, welcher den aktuell angemeldeten Benutzer technisch repräsentierte und bei der Umgebung abgefragt werden konnte.

Im vorangegangenen Kapitel wurde untersucht, welche Anforderungen an ein fachlich motiviertes Modell eines Benutzers bestehen. Diese Anforderungen konnten durch `dvUser` nicht erfüllt werden, da dieser Fachwert lediglich den Namen des Anwenders kapselte. Damit war eine fachliche Identifizierung, nicht jedoch die Modellierung des Privatbesitzes, möglich.

4.2.2. Elemente des fachlichen Benutzermodells

Das fachliche Modell des Benutzers repräsentiert die Eigenschaften des Benutzers im System, die für den softwaretechnischen Entwurf als wesentlich erachtet werden. Hierzu gehören seine fachliche Identität und Privatbesitz, wie in Kapitel 3 beschrieben wurde.

Der Benutzer führt den Privatbesitz immer mit sich, egal in welcher Arbeitsumgebung er gerade arbeitet. Bei dem Privatbesitz handelt es sich um die Modelle anwendungsfachlicher Gegenstände, wie z.B. Terminkalender, Schlüssel und Zertifikate. Es liegt daher nahe, diese Ge-

genstände in fachlichen Behältern aufzubewahren, die dann zum fachlichen Modell des Benutzers gehören. Im Rahmen dieser Arbeit sind für die Umsetzung in den Entwurf als Privatbesitz lediglich Zertifikate und Schlüssel relevant. Dabei werden Schlüssel in Anlehnung an den Anwendungsbereich an einem Schlüsselbund aufbewahrt, der als fachlicher Behälter modelliert wird. Die Verwaltung anderer Gegenstände im Privatbesitz des Anwenders wird in dieser Arbeit weder diskutiert noch modelliert. Als offene Frage verbleibt:

> **Offene Frage 4**
> Wie können Gegenstände des Privatbesitzes allgemein im Rahmen des fachlichen Benutzermodells gehandhabt werden?

Die fachliche Identität eines Anwenders wird im Anwendungsbereich durch seinen Namen in einem bestimmten Kontext (z.B. einer Abteilung) sowie gegebenenfalls durch seine Personalnummer sichergestellt. Aus technischer Sicht ist dies nicht hinreichend, da Namen mehrfach vorkommen und sich ändern können. Des weiteren ist der Kontext, in dem der Name verwendet wird, in Anwendungssystemen potentiell wesentlich größer. Anwendungssysteme können die Zusammenarbeit sehr vieler Anwender unterstützen, die sich nicht zwangsläufig jemals persönlich begegnen werden. Dieses Problem wird durch eindeutige, für Menschen lesbare Benutzerkennungen gelöst, deren Verwendung im Anwendungssystem obligatorisch ist.

> **Ergebnis 23**
> Das fachliche Benutzermodell enthält eine Benutzerkennung. Diese ist für Menschen lesbar und bezeichnet einen Anwender eindeutig.

Wie in Abschnitt 4.1.5 beschrieben wurde, werden im JWAM-Rahmenwerk häufig keine technischen Referenzen, sondern fachliche Verweise verwendet, wenn Gegenstände in einer fachlichen Verbindung zueinander stehen (z.B. ein Konto und der Kontoinhaber). Zur Beibehaltung einer einheitlichen Handhabung von fachlichen Verweisen soll von diesem Prinzip nicht abgewichen werden. Zudem sind in diesem Fall keine konzeptionellen Änderungen in den Rahmenwerks-Komponenten notwendig, die bereits den Fachwert `dvUser` verwenden.

Zur technischen Identifikation ist ein `dvIdentificator` ist geeignet. Ungeeignet ist dieser hingegen, den Benutzer wie eine Benutzerkennung zu repräsentieren. Ein fachlicher Verweis zum Referenzieren von Benutzern soll daher im softwaretechnischen Modell neben einem `dvIdentificator` die Benutzerkennung beinhalten. Er ersetzt den bisher im Rahmenwerk vorhandenen Fachwert `dvUser`.

Das fachliche Modell des Benutzers modelliert ausgewählte Eigenschaften eines Benutzers der Anwendungswelt. Diese Eigenschaften können sich ändern (z.B. der Benutzername durch eine Heirat), so daß das fachliche Modell angepaßt werden muß. Aus Sicht des WAM-Ansatzes wird das Benutzermodell daher als Material betrachtet, welches bei Bedarf mit einem Werkzeug bearbeitet wird (siehe Abschnitt 4.2.4).

> **Ergebnis 24**
> Die fachlichen Benutzermodelle können als Materialien betrachtet werden. Zu ihrer fachlichen Identifizierung wird ein Fachwert verwendet, der die Benutzerkennung zur Repräsentation eines Anwenders sowie einen `dvIdentificator` zur Identifikation beinhaltet.

4.2.3. Ein Standard-Modell eines Benutzers

WAM-Anwendungssysteme werden im allgemeinen als verteilte Mehrbenutzersysteme entworfen. Gleichwohl gibt es auch Situationen, in denen Einzelplatzsysteme benötigt werden. In diesem Fall ist die Unterscheidung verschiedener Anwender häufig irrelevant, und es kann aus technischer Sicht ein Modell eines Standard-Benutzers verwendet werden, wenn ein Benutzermodell benötigt wird.

4.2.4. Verwaltung von Benutzern

In Anwendungssystemen, die von mehreren Anwendern verwendet werden, existiert eine entsprechend große Anzahl von fachlichen Benutzermodellen. Wenn ein Anwender hinzukommt oder die Organisation verläßt, müssen die entsprechenden Modelle hinzugefügt bzw. gelöscht oder archiviert werden. Teilweise ist es auch notwendig, ein existierendes Benutzermodell zu verändern (s.o.). Diese Aufgaben werden mit einem Werkzeug zur Benutzerverwaltung durchgeführt. Dieses Werkzeug arbeitet auf einem Modell der Menge der Benutzer, welches in dieser Arbeit als *Benutzerorganisation* bezeichnet wird, da es die Organisation der Benutzer des Anwendungsbereichs im Anwendungssystem widerspiegelt.

Bei der Benutzerorganisation handelt es sich um eine Komponente mit Dienstleister-Charakter: sie stellt Informationen (die Benutzermodelle) zentral bereit, löst fachliche Verweise auf Benutzer auf und kümmert sich um deren Speicherung in einem Persistenzmedium. Der Zugriff auf die Benutzerorganisation erfolgt aus der Umgebung eines Arbeitsplatzes heraus.

> **Ergebnis 25**
> Die fachlichen Benutzermodelle werden in einem fachlichen Service namens Benutzerorganisation organisiert.

Am Rande dieser Betrachtungen sei bemerkt, daß es sich bei der Verwaltung und Zurverfügungstellung von Benutzermodellen aktuell um ein viel diskutiertes Thema handelt. Speziell größere Organisationen haben das Problem, diese Modelle in großem Maßstab zu verwalten und aktuell zu halten. Als technische Lösung hierfür werden sogenannte *Directory Server* verwendet. Hierbei handelt es sich im Prinzip um Lesezugriff-optimierte Datenbanken, die an verschiedenen Standorten aufgestellt werden und sich automatisch untereinander synchronisieren. Anwendungssysteme können über ein spezielles Protokoll auf diese Server zugreifen. Als Industrie-Standard hierfür hat sich das *Lightweight Directory Access Protocol* (LDAP) durchgesetzt. Zur persistenten Datenhaltung der Benutzerorganisation wären solche LDAP-fähigen Server geeignet (vgl. Abbildung 28). Aufgrund des Umfang dieses Themengebietes können diese Betrachtungen im Rahmen dieser Arbeit nicht vertieft werden. Als offene Frage wird festgehalten:

> **Offene Frage 5**
> Wie können Directory Server im Zusammenhang mit der Benutzerorganisation zur Speicherung der fachlichen Benutzermodelle verwendet werden?

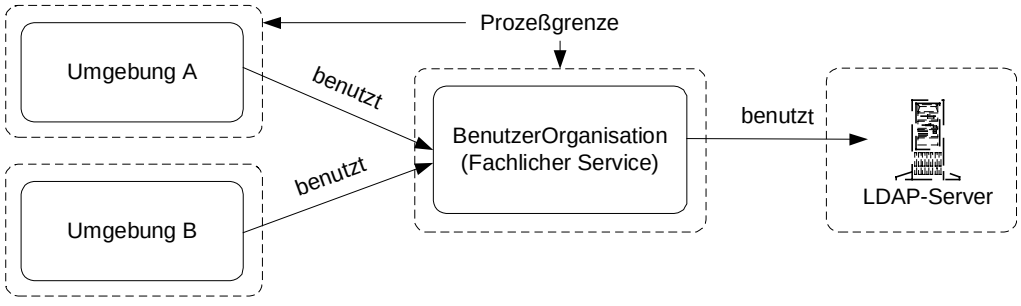


Abbildung 28: Zugriff auf die Benutzerorganisation

4.2.5. Anmelden eines Benutzers

Wenn sich ein Anwender im System anmelden möchte, muß seine Identität verifiziert werden. Wie in Kapitel 2 dargestellt wurde, gibt es verschiedene Verfahren zur Zugangskontrolle. Die Betrachtung von Zugangskontrollsystemen ist kein zentrales Thema dieser Arbeit. Daher wird vorgeschlagen, zur Zugangskontrolle im Rahmenwerk den wissensbasierten Ansatz (siehe Abschnitt 2.2.1) zu verwenden, da dieser am einfachsten zu realisieren ist. Als Geheimnis, welches der Benutzer kennen muß, wird ein Paßwort verwendet. Dieses Paßwort wird damit

zum Bestandteil des fachlichen Modells des Benutzers. Nach erfolgreicher Anmeldung wird der Benutzer in der Umgebung registriert. Dort kann der aktuell angemeldete Benutzer abgefragt bzw. ein neuer Benutzer registriert werden. Des weiteren wird sein persönlicher Arbeitsplatz geladen und dargestellt.

4.2.6. Zusammenfassung und Ausblick auf die Konstruktion

Beim Entwurf des softwaretechnischen Benutzermodells werden folgende Aspekte berücksichtigt: der Name des Benutzers sowie eine Benutzerkennung, ein fachlicher Behälter für Zertifikate und Schlüssel, ein fachlicher Verweis zur Identifikation und Repräsentation sowie ein Paßwort zu Authentifizierung. Weiterhin soll ein Standard-Benutzermodell existieren.

Das fachliche Modell des Benutzers sowie eine vereinfachte Benutzerorganisation wurden im Rahmen dieser Arbeit konstruiert und in die Umgebung das JWAM-Rahmenwerkes integriert. Eine Beschreibung der Konstruktion ist in Kapitel 5 zu finden. Die Personalisierung der Arbeitsumgebung konnte nicht konstruktiv umgesetzt werden, da sie sich als zu umfangreich erwies.

4.3. Umsetzung der Schlüssel-Schloß-Metapher

Im Rahmen der Beschreibung der Umsetzung der Schlüssel-Schloß-Metapher wird zunächst ein fachliches Modell eines Schlüsselbartes sowie seine softwaretechnische Umsetzung vorgestellt. Das Schlüsselbart-Modell ist zwar fachlich motiviert, trägt jedoch zum Verständnis der Schlüssel-Schloß-Metapher selbst nicht bei und wurde daher in Kapitel 3 nicht beschrieben. Bei der Umsetzung in des softwaretechnische Modell wird der Schlüsselbart hingegen berücksichtigt.

4.3.1. Der Schlüsselbart

Jeder Schlüssel verfügt über Merkmale, so daß er zu einem Schloß paßt. Obwohl es inzwischen auch Schlüssel mit elektronischen (z.B. Chips) oder magnetischen Merkmalen gibt, handelt es sich bei mechanischen Schlüsseln meist um Bohrungen am Bart (engl. *Keybit*) eines Schlüssels (siehe Abbildung 29). Wenn die Merkmale zu einem Schloß passen, kann der Schlüssel das Schloß öffnen. Über die Merkmale werden auch hierarchische Beziehungen codiert: ein Schlüsselbart eines Sub-Schlüssels ist eine Spezialisierung eines oder mehrerer Super-Schlüssel (vgl. Abschnitt 3.3.1). Zwei Schlüsselbärte sind dann gleich, wenn sie dieselben Merkmale besitzen – unabhängig davon, zu welchem Schlüssel sie gehören. Wird der Bart eines Schlüssels mechanisch verändert, handelt es sich immer noch um denselben Schlüssel, nicht jedoch um denselben Bart. Der veränderte Bart paßt nicht mehr in dasselbe Schloß, und im eigentlichen Sinne handelt es sich um einen anderen Bart.

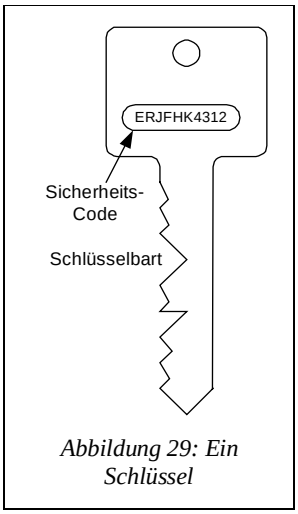


Abbildung 29: Ein Schlüssel

In Kapitel 3 wurde darauf hingewiesen, daß im Rahmen der Schlüssel-Schloß-Metapher Sicherheitsschlüssel verwendet werden. Eine Eigenschaft eines Sicherheitsschlüssels ist, daß er mit einem Sicherheitscode beschriftet ist. Dieser Code identifiziert ihn zum einen als Sicherheitsschlüssel und zum anderen kann der Hersteller anhand dieses Codes das zugehörige Zertifikat ermitteln. In gewissem Sinne drücken Schlüsselbart und Sicherheitscode demnach dasselbe aus: bei beiden handelt es sich um Merkmale eines Schlüssels, mit deren Hilfe überprüft werden kann, ob ein Schlüssel zu einem Schloß paßt.

Schlüsselbärte sind ein wesentlicher Bestandteil mechanischer Schlüssel und werden im softwaretechnischen Modell eines Schlüssels berücksichtigt. In der Anwendungswelt hat ein Schlüsselbart Materialcharakter. Er wird allerdings nur gebraucht, um mit dem ebenfalls mechanischen Gegenstand „Schloß“ umgehen zu können. Der gleichsam bedeutsame Sicherheitscode kann mechanisch von einem Schloß nicht erfaßt werden.

Bei der Übertragung in das softwaretechnische Modell stellt sich heraus, daß der Bart eines Schlüssels weniger Objekt- und mehr Wertcharakter besitzt. So kann er z.B. nicht unter Wahrung seiner Identität verändert werden (vgl. Abschnitt 4.1.4). Noch deutlicher wird dieser Wertcharakter bei elektronisch basierten Schlüsseln (z.B. in Form einer Chipkarte). Hier existiert gar kein mechanisches Merkmal mehr, statt dessen wird ein geheimgehaltener Code verwendet, mit dessen Hilfe die Karte als gültig verifiziert werden kann (vgl. [Beis, Vorwerk 97]). Aus diesem Grund wird ein Schlüsselbart im softwaretechnischen Modell als Fachwert modelliert. Der Fachwert beinhaltet ein individuelles Merkmal, welches ihn von anderen Schlüsselbärten unterscheidet.

Ergebnis 26
Schlüsselbärte können als Fachwerte modelliert werden.

Ebenfalls Bestandteil des Modells eines Schlüsselbartes ist der Sicherheitscode. Dieser dient in der Anwendungswelt im wesentlichen zur Identifikation des zugehörigen Zertifikates und wird daher durch einen `dvIdentificator` modelliert. Die hierarchischen Strukturen, die durch den Schlüsselbart ausgedrückt werden können (s.o.), werden durch einen zusammengesetzten Fachwert modelliert: ein Sub-Schlüsselbart-Fachwert umhüllt seine Super-Schlüsselbart-Fachwerte und beinhaltet zur Spezialisierung ein zusätzliches individuelles Merkmal (siehe Abbildung 30).

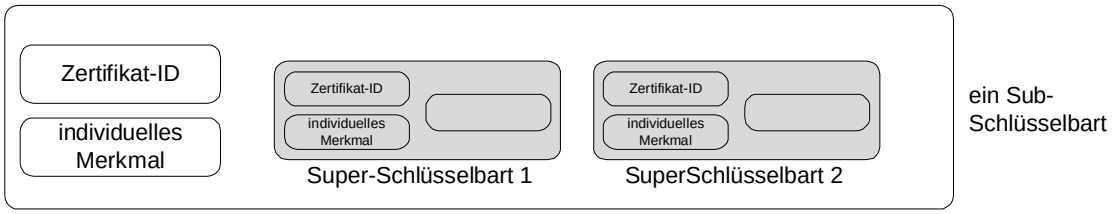


Abbildung 30: Konstruktion von Schlüsselbärten mit Fachwerten

4.3.2. Modellierung von Schlüsseln, Schlössern und Zertifikaten

Aus der Definition von Schlüsseln, Schlössern und Zertifikaten geht hervor, daß es sich sowohl im Anwendungsbereich als auch in der Metapher um Gegenstände handelt. Daher sollen Schlüssel, Schlösser und Zertifikate auch im softwaretechnischen Entwurf als Gegenstände modelliert werden, wie sie in Abschnitt 4.1.5 beschrieben wurden.

Schlüssel modellieren die Zugriffsrechte von Anwendern und können hierarchisch organisiert sein. Welches Zugriffsrecht ein Schlüssel darstellt, wird durch seinen Schlüsselbart ausgedrückt. In einem mechanischen Schloß im Anwendungsbereich ist ein Mechanismus (der Schließzylinder) hinterlegt, mit dem das Merkmal des Schlüssels (der Schlüsselbart) überprüft wird. Sind die Merkmale gleich, oder ist das Merkmal des Schlosses eine Spezialisierung des Merkmals des Schlüssels, paßt der Schlüssel zu dem Schloß und kann es auf- oder zuschließen. Dieser Mechanismus kann einfach in das softwaretechnische Modell übertragen werden, indem das Schloß einen Schlüsselbart enthält. Durch Vergleich der Schlüsselbart-Fachwerte von Schloß und Schlüssel erfährt man, ob der Schlüssel zu dem Schloß paßt.

Hierbei handelt es sich im softwaretechnischen Modell um eine Vereinfachung gegenüber dem Anwendungsbereich. Das Überprüfen eines Schlüsselbartes durch einen Zylinder ist erheblich komplizierter. Diese Komplexität ist jedoch durch die mechanische Struktur des Schlüssels bestimmt und damit im Anwendungsbereich technisch und nicht fachlich motiviert. Sie kann im softwaretechnischen Modell vereinfacht werden.

Ergebnis 27
Schlüssel, Schlösser und Zertifikate werden als Gegenstände modelliert. Schlüssel und Schlösser beinhalten das Modell eines Schlüsselbartes. Paßt der Bart eines Schlüssels zu dem eines Schlosses, kann der Schlüssel das Schloß abschließen.

In der Metaphernbeschreibung wurde ein Zertifikat als Gegenstand definiert, der die Eigentümerschaft über bestimmte Schlüssel und Schlösser ausdrückt. Auf dem Zertifikat findet sich eine Identifikationsnummer sowie der Name des Eigentümers. Das Zertifikat wird in dieser Form in den softwaretechnischen Entwurf übernommen.

4.3.3. Erzeugen von Schlüsseln, Schlössern und Zertifikaten

Schlüssel, Schlösser und Zertifikate werden durch einen fachlichen Service erzeugt (vgl. Abschnitt 3.3.4). Bei diesem Service ist hinterlegt, welcher Schlüsselbart zu einem Zertifikat gehört. Der Service speichert Informationen über alle Zertifikate, Schlüssel und Schlösser, die er herausgegeben hat. Auf diese Weise kann nachvollzogen werden, welcher Anwender wann einen Gegenstand erzeugen ließ. Anwender nehmen die Dienste des Service in Anspruch, indem sie ein Werkzeug benutzen, das diesen Service verwendet.

Benutzt ein Anwender den Service um ein neues Zertifikat zu erzeugen, generiert der Service zunächst einen Schlüsselbart-Fachwert. Dieser wird zusammen mit der Information gespeichert, wer das Zertifikat erzeugt hat. Anschließend gibt der Service das Zertifikat heraus und das zugehörige Werkzeug fügt es dem Privatbesitz des Benutzers hinzu. Soll ein Sub-Zertifikat erzeugt werden, müssen zusätzlich Super-Zertifikate übergeben werden.

Will ein Anwender mit einem Zertifikat einen Schlüssel (oder ein Schloß) erzeugen, wendet er sich mit dem Zertifikat an den Service. Wenn der Service sichergestellt hat, daß der Anwender der Eigentümer des Zertifikates ist, sucht er anhand der Identifikationsnummer des Zertifikates den Schlüsselbart-Fachwert heraus und erzeugt auf dieser Basis den gewünschten Schlüssel (bzw. das Schloß). Auch bei dieser Anfrage wird protokolliert, welcher Anwender den Service in Anspruch genommen hat.

4.3.4. Ausblick auf die Konstruktion

Schlüssel, Schlösser und Zertifikate sowie der fachliche Service, der für ihre Herstellung zuständig ist, wurden im Rahmen dieser Arbeit für das JWAM-Rahmenwerk konstruiert und integriert (siehe Kapitel 5). Ein Werkzeug zur Bedienung des Service konnte nicht gebaut werden, da sich dessen Konstruktion als zu umfangreich erwies.

4.4. Verschließbare Gegenstände

In Abschnitt 4.1.5 wurde eine Begriffshierarchie der Entwurfskonzepte vorgestellt, wobei auf die Gegenständlichkeit der durch Entwurfsmetaphern modellierten Dinge hingewiesen wurde. Für die Konstruktion einer Zugriffskontrolle erweist sich das Gegenstandskonzept als erheblicher Vorteil, denn im softwaretechnischen Modell kann nun prinzipiell über den Schutz von Behältern, Werkzeugen usw. diskutiert werden. Wenn man z.B. einmal festgestellt hat, wie man Behälter schützen will, muß man nicht mehr prinzipiell über den Schutz spezieller Behälter diskutieren. Statt dessen können die allgemeinen Schutzmechanismen für Behälter angewendet und bei Bedarf verfeinert werden.

Im folgenden wird ein Konzept erarbeitet, mit dem sich insbesondere Behälter und Werkzeuge mit Hilfe von Schlössern im softwaretechnischen Modell absichern lassen. Entsprechende Betrachtungen für fachliche Services und Automaten würden den Rahmen dieser Arbeit sprengen. Als offene Frage wird deshalb festhalten:

> **Offene Frage 6**
> Wie können fachliche Services und Automaten im softwaretechnischen Modell mit Schlössern abgesichert werden?

Als Ansätze zur Beantwortung dieser Frage können gleichwohl das im folgenden herausgearbeitete Konzept sowie die Betrachtung verteilter Ressourcen in Abschnitt 4.7 dienen.

In Bezug auf Behälter und Werkzeuge wurde in Kapitel drei festgehalten, daß diese Gegenstände generell nicht benutzbar sein sollen, wenn der Benutzer keine Zugriffsrechte für sie

hat. Im softwaretechnischen Modell läßt sich dies prinzipiell auf zwei Weisen realisieren: entweder können die Objekte nicht erreicht werden, oder man kann sie nicht benutzen.

4.4.1. Wegschließen von Gegenständen durch Wrapper

Eine Lösung für den ersten Fall ist, das Objekt in ein anderes Objekt einzubetten und dieses Objekt abzuschließen (siehe Abbildung 31). Dies würde bedeuten, ein umhüllendes Objekt (einen sogenannten *Wrapper*, in [Gamma et al. 96] beschrieben als *Adapter*) zu konstruieren, an dem ein Schloß angebracht ist. Der umhüllte Gegenstand kann nur herausgeben werden, wenn das Schloß offen ist. Dies wird durch eine Vorbedingung sichergestellt. Ein Klient[20] bekommt dann über den Wrapper Zugriff auf den Gegenstand.

Der Vorteil diese Ansatzes ist, daß an dem Gegenstand selbst nichts verändert werden muß. Als nachteilig erweist sich allerdings, daß an dem Wrapper keine Informationen über den umhüllten Gegenstand abgelesen werden können. Weder ist bekannt, um welchen Gegenstand es sich handelt, noch können seine Repräsentation oder sein Typ erfragt werden. Gleiches gilt für möglicherweise vorhandene fachliche Operationen. Diese Nachteile ließen sich beheben, indem die Schnittstelle des Wrappers entsprechend erweitert wird. Dies würde allerdings dazu führen, daß man zu einer Vielzahl von anwendungsfachlich motivierten Gegenständen im System entsprechende Wrapper konstruieren müßte.

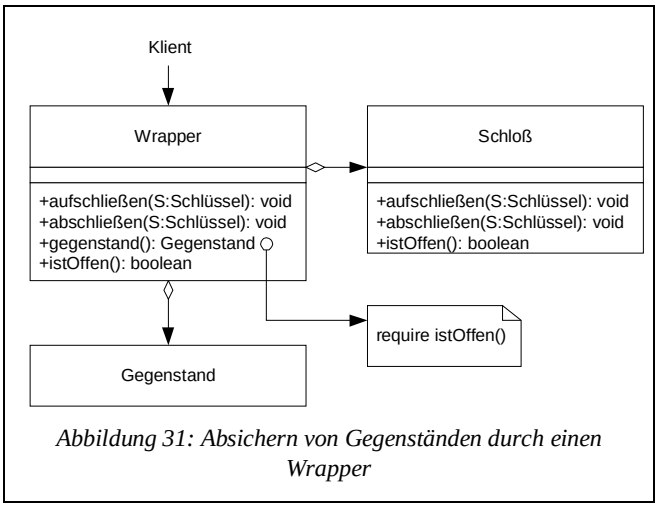


Abbildung 31: Absichern von Gegenständen durch einen Wrapper

Sobald der Wrapper einmal aufgeschlossen war, kann ihn jedes Objekt im System technisch referenzieren. Auch wenn der Wrapper wieder abgeschlossen wird, können diese Referenzen weiter bestehen. So kann es (siehe Abbildung 32) zu dem Zustand kommen, daß ein Werkzeug auf einem umhüllten Behälter nicht mehr arbeiten kann, während ein weiteres Werkzeug noch auf demselben Behälter arbeitet. Das Werkzeug merkt nicht einmal, daß der Wrapper wieder abgeschlossen wurde. Dies entspricht nicht mehr dem Bild eines abgeschlossenen Gegenstandes.

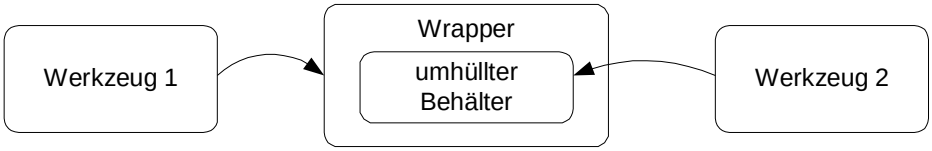


Abbildung 32: Problem bei Verwendung eines Wrappers

Man kann nicht garantieren, daß auf einem Behälter in einem abgeschlossenen Wrapper keine Werkzeuge mehr arbeiten. Diese Werkzeuge könnten (wenn sie unsauber implementiert sind) ihre technische Referenz auf den Behälter sogar an weitere Werkzeuge weitergeben. Der Schutz durch den Wrapper ist damit letztlich wirkungslos.

> **Ergebnis 28**
> Abschließbare Wrapper sind zum Schutz von Gegenständen nicht geeignet. Zum einen ist es kompliziert, geeignete Wrapper zu konstruieren. Zum anderen kann auch ein abgeschlossener Wrapper nicht garantieren, daß der umhüllte Gegenstand nicht benutzt wird.

[20] Analog zum Sprachgebrauch in [Züllighoven et al. 98] wird ein Objekt, daß die Dienstleistung eines anderen Objektes in Anspruch nimmt, als Klient bezeichnet.

4.4.2. Abschließen von Gegenständen

Eine andere Lösung ist, das Schloß zum Bestandteil des Gegenstandes zu machen. Wenn das Schloß abgeschlossen ist, kann der Gegenstand nicht benutzt werden. Dies wird ebenfalls durch Vorbedingungen sichergestellt, im Unterschied zum Wrapper-Konzept handelt es sich aber um Vorbedingungen des Gegenstandes selbst. Nur auf diese Weise kann man erreichen werden, daß der Gegenstand nicht benutzt wird, wenn er abgeschlossen ist. Exemplarisch wird dies für einen Behälter in Abbildung 33 dargestellt. Der Änderungsaufwand im Rahmenwerk, den diese Lösung erfordert, ist moderat. Die Absicherung von Operationen durch Vorbedingungen kann auf Ebene der Behälter-(bzw. Werkzeug-) Oberklasse vorgenommen werden. Diese Möglichkeit besteht bei der Verwendung eines Wrappers nicht.

Eine Variante zu dieser Lösung wäre, ein von „Behälter“ abgeleitete Unterklasse „verschließbarer Behälter“ zu bilden, in die das Schloß integriert wird. Die Unterklasse implementiert dann die Vorbedingungen und leitet den Operationsaufruf an die Oberklasse weiter. Diese Variante kann allerdings grundsätzlich nicht verwendet werden, da auf diese Weise die Vorbedingungen von Operationen verschärft würden. Das Verschärfen von Vorbedingungen in Subklassen ist aber verboten (siehe [Meyer 97], Seite 573): es führt zu Problemen, wenn ein Klient auf der Schnittstelle der Oberklasse arbeitet, da er nicht wissen kann, daß die Vorbedingung einer Operation verschärft wurde[21].

Die bisher vorgestellte Lösung ist tragfähig, allerdings müßten die Schlösser sowohl in die Werkzeug- als auch in die Behälter-Basisklasse integriert werden. Da Werkzeuge und Behälter Gegenstände sind, lassen sich als weitere Abstraktion „verschließbare Gegenstände“ einführen (siehe Abbildung 34). Auf diese Weise wird auch deutlich zum Ausdruck gebracht, daß es neben verschließbaren auch nicht verschließbare Gegenstände gibt. Ein verschließbarer Gegenstand kapselt ein Schloß und bietet Operationen zum Umgang mit verschließbaren Gegenständen an (z.B. „aufschließen“, in Abbildung 34 nicht dargestellt). Das Schloß eines abgeschlossenen Gegenstandes kann nur entfernt und ausgetauscht werden, wenn es offen ist.

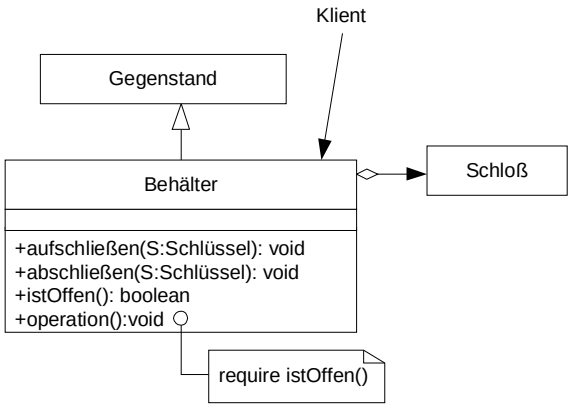


Abbildung 33: Absichern von Behältern durch Anbringen eines Schlosses

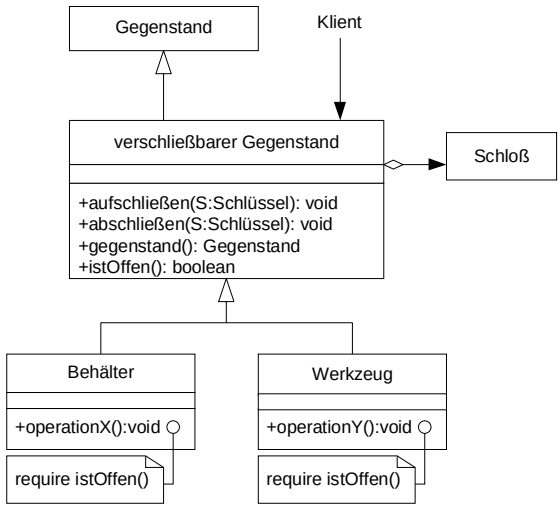


Abbildung 34: Verschließbare Gegenstände

Ergebnis 29
Verschließbare Gegenstände können mit einem Schloß versehen und abgeschlossen werden. Diese Funktionalität wird durch eine Oberklasse zur Verfügung gestellt. Behälter und Werkzeuge können konzeptionell als verschließbare Gegenstände betrachtet und von dieser Oberklasse abgeleitet werden.

[21] Anmerkung: Aus diesem Grund kann auch nicht das *Dekorierer*-Entwurfsmuster aus [Gamma et al. 96] verwendet werden.

Alternativ könnten *mixin classes* ([Booch et al. 99], Seite 142) verwendet werden. Hierbei vereint eine Unterklasse die Eigenschaften zweier Oberklassen. Abgesehen davon, daß Mehrfachvererbung generell kritisch betrachtet wird und umsichtig benutzt werden sollte (vgl. [Booch et al. 99], [Meyer 97]), steht sie in Java (und damit im JWAM-Rahmenwerk) nicht zur Verfügung. Dieser Ansatz wird daher nicht näher betrachtet.

Es sei darauf hingewiesen, daß ein verschließbarer Gegenstand ein Schloß haben kann, aber nicht haben muß. Die Operation istOffen() gibt immer `true` zurück, wenn der Gegenstand über kein Schloß verfügt. Ein Gegenstand ohne Schloß ist daher nicht durch Zugriffskontrollen geschützt. Welche Auswirkung ein abgeschlossenes Schloß auf einen verschließbaren Gegenstand hat, kann nur im speziellen Fall entschieden werden (siehe Abschnitt 3.4). Es müssen nicht notwendigerweise alle Operationen eines speziellen Gegenstandes istOffen() als Vorbedingung haben. Das Aufweichen einer Vorbedingung ist durchaus legitim (siehe wiederum [Meyer 97], Seite 573).

4.4.3. Abschließen von Subwerkzeugen

Bei der Modellierung von Softwarewerkzeugen werden konzeptionelle Eigenschaften von Werkzeugen der Anwendungswelt übernommen, allerdings ist eine „direkte Abbildung der Handhabung und Gestalt selten sinnvoll“ ([Züllighoven et al. 98], Seite 84). Bei Softwarewerkzeugen handelt es sich meist um komplexere Werkzeuge, als sie in der Anwendungswelt vorgefunden werden. Solche komplexen Werkzeuge sind häufig aus Subwerkzeugen zusammengesetzt und unterstützen mehrere Aufgaben aus einem Arbeitszusammenhang (vgl. Abschnitt 4.1.7).

Möglicherweise darf nicht jeder Anwender alle diese Aufgaben erledigen und benötigt daher die entsprechenden Werkzeugteile nicht. In diesem Fall ist es sinnvoll, die Werkzeugteile abzuschließen, die nicht benutzt werden sollen. Damit läßt sich ein Werkzeug dynamisch den speziellen Bedürfnissen des Kunden und den Aufgaben und Kompetenzen des Anwenders anpassen – der Funktionsumfang eines Werkzeugs ist dann abhängig vom jeweiligen Benutzer (vgl. [Züllighoven et al. 98], Seite 183).

Beim Entwurf eines Werkzeugs stellt sich die Frage, wie feingranular einzelne Funktionen von Werkzeugen abgeschlossen werden können. In der Konstruktion ließe sich dies bis auf die Ebene einzelner Methoden anwenden. Je mehr das Werkzeug zur Unterstützung klar definierter Aufgaben entworfen wird, desto deutlicher ist seine Gliederung in Subwerkzeuge. Diese können mit Schlössern und der Anwender mit entsprechenden Schlüsseln ausgestattet werden. Fehlt ihm ein Schlüssel, macht sich dies z.B. durch ein fehlendes oder ausgegrautes Subwerkzeug bemerkbar.

> **Ergebnis 30**
> Subwerkzeuge sind abschließbar. Das umgebende Werkzeug entscheidet, wie es mit einem abgeschlossenen Subwerkzeug umgeht.

Die beschriebene Lösung stellt auf den ersten Blick einen Bruch mit der ursprünglichen Werkzeugmetapher dar. Dasselbe Werkzeug kann sich verschiedenen Anwendern gegenüber unterschiedlich präsentieren. Ebenso kann es sich einem Anwender zu verschiedenen Zeitpunkten anders präsentieren. Ein solches Verhalten ist bei einfachen Werkzeugen (z.B. einem Messer) nicht zu beobachten. Es wurde allerdings bereits darauf hingewiesen, daß Softwarewerkzeuge nicht einfache Werkzeuge im Anwendungsbereich nachbilden, sondern sich bereits im Ansatz durch ein mehr an Komplexität von ihnen unterscheiden. Die Werkzeugmetapher läßt sich daher durchaus in der beschriebene Weise interpretieren.

Die Abschließbarkeit von Subwerkzeugen ist darüber hinaus auch technisch motiviert. Würde man aber für jede mögliche Kombination von Subwerkzeugen ein eigens Werkzeug bauen, erhielte man eine Vielzahl von Werkzeugen mit ähnlicher fachlicher Funktionalität. Aus Sicht des Anwenders wäre der Effekt derselbe, als ob er nur ein Werkzeug erhielte. Für die Syste-

mentwickler und Systembetreuer bedeutet dies jedoch einen erheblichen Mehraufwand, da der Anwender immer ein passendes Werkzeug zur Verfügung gestellt bekommen muß. Wenn Werkzeuge aber den Aufgaben der Anwender entsprechend in Subwerkzeuge gegliedert sind, können die Anwender die entsprechenden Schlüssel einfach vom Systembetreuer erhalten.

4.4.4. Technischer Umgang mit verschließbaren Gegenständen

Durch das Anbringen von Schlössern an Werkzeuge und Behälter ergeben sich Konsequenzen für alle technischen Komponenten, die mit Werkzeugen und Behältern umgehen. Da der Zugriffsschutz durch eine Vorbedingung auf Methodenebene realisiert wird, muß darauf geachtet werden, daß die geschützten Operationen nicht aufgerufen werden, da sonst eine Exception ausgelöst wird (vgl. Abschnitt 4.1.9).

Für Behälter bedeutet dies im wesentlichen, daß Werkzeuge, die auf ihnen arbeiten, entsprechend angepaßt werden müssen. Sie müssen so entworfen sein, daß sie mit ganz oder teilweise verschlossenen Behältern arbeiten können. Operationen, deren Aufruf zu einer Exception führen würde, werden nicht aufgerufen. Der Anwender wird durch eine entsprechende Repräsentation des Behälters darüber informiert, daß er bestimmte fachliche Funktionen nicht nutzen kann. Nach Möglichkeit stellt die Umgebung den Behälter so dar, daß für den Anwender offensichtlich ist, falls er ihn nicht verwenden kann.

Abgeschlossene Werkzeuge dürfen nicht gestartet werden. Für einfache, nicht eingebettete Werkzeuge sowie für Kontextwerkzeuge bedeutet dies, daß ein technischer Eingriff in die Umgebung notwendig ist. Diese muß das Starten abgeschlossener Werkzeuge unterbinden und dem Anwender – wie bei Behältern – visualisieren, daß er das Werkzeug nicht verwenden kann. Bei eingebetteten Subwerkzeugen muß hingegen das umgebende Werkzeug darauf achten, daß das abgeschlossene Subwerkzeug nicht gestartet wird. Wie ein abgeschlossenes Subwerkzeug dargestellt wird, entscheidet das umgebende Werkzeug. Beispielsweise könnte das Werkzeug gar nicht dargestellt und der entsprechende Platz im Kontextwerkzeug leer gelassen oder auf andere Weise gefüllt werden.

Darüber hinaus ist zu klären, wie Schlösser an Behälter und Werkzeuge angebracht werden können. Die Frage ist nur vor einem konkreten konstruktiven Hintergrund zu beantworten. Eine Lösung für das JWAM-Rahmenwerk wird in Kapitel 5 vorgestellt.

4.4.5. Ausblick auf die Konstruktion

Das hier vorgeschlagene Konzept zur Absicherung von Werkzeugen und Automaten ist im Rahmen dieser Arbeit konstruiert und in das JWAM-Rahmenwerk integriert worden. Eine Beschreibung der Konstruktion findet sich in Kapitel 5.

4.5. Protokollierung von Zugriffen

Die Zurechenbarkeit von Transaktionen (accountability) ist eine zentrale Forderung von Sicherheitsrichtlinien (siehe Kapitel 2). Die dabei entstehenden Protokolle müssen den Regeln des Datenschutzes entsprechend aufbewahrt und ausgewertet werden.

Bei der Ausführung sicherheitsrelevanter Funktionen kann eine Protokollierung des Zugriffes erfolgen. Hierdurch übernehmen die Benutzer des Systems nachvollziehbar Verantwortung für die von ihnen ausgeführten Systemfunktionen. Auch wenn ein Anwender eine Funktion im Normalfall ohne Protokollierung nutzen darf, ist ihre Ausführung bei der Wahl bestimmter Parameter möglicherweise an besondere Rechte gebunden und muß protokolliert werden.

> *Beispiel:*
> Ein Bankangestellter führt eine Überweisung aus, bei der Auftraggeber- und Empfängerkonto identisch sind (Eigenüberweisung). Für diese spezielle Überweisung benötigt er einen Schlüssel. Wenn eine Eigenüberweisung ausgeführt wird, muß dies protokolliert werden.

Eine Protokollierung von Zugriffen ist auch während der Einführungsphase eines Systems sinnvoll und hilfreich. Aus den Protokollen kann dann entnommen werden, wo Zugriffskon-

trollen notwendig sind, und welche Anwender Zugriff erhalten müssen. Ein absichtlicher oder unabsichtlicher Mißbrauch neuer Systemkomponenten kann auf diese Weise nicht verhindert werden, aber es ist immerhin möglich, den verantwortlichen Benutzer zu ermitteln.

Das Protokollieren von Zugriffen ist im wesentlichen technisch motiviert und erforderlich, weil Anwender viel häufiger persönlichen Kontakt zueinander haben, wenn sie kein Anwendungssystem verwenden. Wichtige Arbeitsschritte werden dabei immer von mehreren Personen durchgeführt. Durch den persönlichen Kontakt haben die Beteiligten immer einen Eindruck davon, wer welche Aktion verantwortlich ausgeführt hat. Das Fehlen dieser gegenseitigen Aufsicht kann durch Protokolle und ihre Auswertung zumindest zum Teil ersetzt werden. Maßnahmen zur technischen Umsetzung dieses Konzeptes wurden im Rahmen dieser Arbeit nicht behandelt. Es verbleibt als offene Frage:

> **Offene Frage 7**
> Wie können Protokolle über die Verwendung von Werkzeugen und das Ausführen sicherheitsrelevanter Funktionen geführt und im Sinne des Datenschutzes sicher gehandhabt werden?

4.6. Unterstützung anderer Arbeitsplatztypen

In Abschnitt 3.6 wurde beschrieben, daß das Auf- und Abschließen von Gegenständen an Experten- und Gruppenarbeitsplätzen explizit geschehen soll. Weiterhin wurde darauf hingewiesen, daß diese Handhabung für Funktionsarbeitsplätze sowie Selbstbedienungsautomaten nicht geeignet ist. Als Vereinfachung kann die Routinetätigkeit „Schließen“ einem Benutzer solcher Arbeitsplätze (implizit) abgenommen werden, wenn er über einen passenden Schlüssel verfügt:

> *Vision: implizites Schließen auf dem Desktop*
> Der Anwender verfügt über einen passenden Schlüssel zu dem Schloß eines verschlossenen Behälters. Er möchte den Inhalt des Behälters betrachten und zieht das Symbol des Auflisters auf das Symbol des Behälters. Der Behälter wird automatisch aufgeschlossen und geöffnet. Beim Schließen des Auflisters wird das Schloß des Behälters automatisch abgeschlossen.

Wie bereits in Abschnitt 3.6.1 erwähnt wurde, sollen Benutzer von Funktionsarbeitsplätzen sowie Selbstbedienungsautomaten nur Gegenstände präsentiert bekommen, die sie auch verwenden können. Durch die Verwendung von implizitem Schließen ist es möglich, die Existenz einer Zugriffskontrolle vollständig vor dem Anwender zu verbergen. Dies erleichtert den Umgang mit dem Anwendungssystem erheblich. Bei Expertenarbeitsplätzen und Gruppenarbeitsplätzen hingegen ist die Wahrnehmung von Zugriffskontrollen durchaus erwünscht.

Im Rahmen dieser Arbeit konnte ein impliziter Schließmechanismus nicht entworfen und implementiert werden. Als offene Frage verbleibt:

> **Offene Frage 8**
> Wie kann ein impliziter Schließmechanismus entworfen und implementiert werden, mit dem sich insbesondere Funktionsarbeitsplätze und Selbstbedienungsautomaten ausrüsten lassen?

4.7. Zugriffskontrolle und gemeinsame Ressourcen

WAM-Anwendungssysteme werden in den meisten Fällen zur Unterstützung kooperativer Arbeitssituationen entworfen. Entsprechende Kooperationstypen wurden in Abschnitt 3.7 vorgestellt. Dabei wurde deutlich, daß Anwender mit Komponenten arbeiten, die außerhalb ihrer persönlichen, lokalen Arbeitsumgebung angesiedelt sind (z.B. die Registratur). Darüber hinaus gibt es noch weitere Komponenten, die „zentral organisiert sind und mehreren Arbeitsumgebungen zur Verfügung stehen“ (siehe [Otto, Schuler 00]). In Anlehnung an die zitierte Arbeit werden diese Komponenten hier *gemeinsame Ressourcen* genannt:

Definition 36: gemeinsame Ressource
Auf eine *gemeinsame Ressource* kann von mehreren Arbeitsumgebungen aus zugegriffen werden.

Bereits bei der Beschreibung der Registratur wurde deutlich, daß auch diese gemeinsamen Ressourcen durch Zugriffskontrollen geschützt werden müssen. Im folgenden wird daher ein Ansatz vorgestellt, wie Schlüssel und Schlösser für fachliche Zugriffskontrollen auf gemeinsame Ressourcen genutzt werden können. Dieser Ansatz wurde im Rahmen dieser Arbeit nicht konstruktiv umgesetzt.

Beim Zugriff auf eine gemeinsame Ressource entsteht eine gegenüber dem lokalen Zugriff besondere Situation. Gemeinsame Ressourcen können von vielen Benutzern gleichzeitig oder in enger zeitlicher Folge benutzt werden. Schließt ein autorisierter Anwender die Komponente auf, steht sie – nach dem bisherigen Benutzungsmodell – auch allen nicht autorisierten Anwendern offen. Da dieser Effekt nicht gewünscht ist, muß das Auf- und Abschließen von verteilten Komponenten anders gehandhabt werden als von lokalen.

Es wird daher vorgeschlagen, den Zustand eines Schlosses nicht auf die gesamte gemeinsame Ressource, sondern auf eine Verbindung zwischen dieser und der lokalen Arbeitsumgebung zu beziehen. Bei jedem Aufbau einer technischen Verbindung wird dann durch die gemeinsame Ressource überprüft, ob der Anwender einen passenden Schlüssel hat. Nur wenn dies der Fall ist, kann der Anwender die gemeinsame Ressource benutzen. Offen bleibt die Frage, wie dieser Ansatz technisch umgesetzt werden kann:

Offene Frage 9
Wie kann der Schutz gemeinsamer Ressourcen durch die Schlüssel-Schloß-Metapher technisch realisiert werden?

4.8. Abgrenzung gegenüber dem Java-Sicherheitsmodell

Dieser Abschnitt soll einen Überblick über das Java-Sicherheitsmodell des JRE 1.3 (*Java-Runtime-Environment* 1.3) geben. Die Darstellungen beruhen auf [Sun 00a], [Sun 00b], [Felten, McGraw 98], [Flanagan 99] und [Sohr 00] und berücksichtigen Sicherheitsaspekte, die für die Ausführung von Java Code auf einem Arbeitsplatzrechner relevant sind.

4.8.1. Allgemeines

Grundvoraussetzung für den Ablauf von Java-Programmen ist das *Java-Runtime-Environment* (JRE), welches im wesentlichen aus den Klassen des Java Kerns und einer *Java Virtual Machine* (VM) besteht (siehe [Sun 96]). Die VM interpretiert den *Bytecode*, welchen der Java-Compiler aus Java-Quellcode erzeugt.

Java Programme werden hauptsächlich in verteilten Anwendungssystemen und insbesondere im Internet (in Form von Applets) als Bestandteil von Webseiten eingesetzt. Dabei kann auch Java-Code verwendet werden, der auf dem lokalen Arbeitsplatzrechner nicht zur Verfügung steht. Dieser wird dann von einer angegebenen Quelle heruntergeladen und in der VM ausgeführt. Besonders bei Webseiten stellt dies ein Problem dar, weil ohne Wissen des Anwenders fremder Java-Code ausgeführt werden kann. Dies ist vom Standpunkt der Sicherheit her bedenklich, weil dieser Code potentiell erhebliche Schäden anrichten kann. Diesen Problemen begegnet das JRE mit einem Sicherheitsmodell, daß laut [Sun 00b] aus den folgenden vier Komponenten besteht:

- Das Sprachdesign verhindert viele mögliche Programmierfehler (z.B. unerlaubte Typumwandlungen).
- Java-Code wird nicht in Maschinensprache, sondern in einen plattformunabhängigen Bytecode übersetzt. Der Bytecode-Verifizierer stellt sicher, daß der Bytecode im Sinne der Bytecode-Spezifikation korrekt ist.

- Der Klassenlader (engl. *class loader*) ist dafür zuständig, den Bytecode von Java-Klassen aus einer Ressource (meist Dateisystem oder Internet) heraus zu laden, damit dieser durch die VM ausgeführt werden kann. Unter anderem sorgt er dafür, daß Klassen aus verschiedenen Quellen unterschiedliche Namensräume erhalten und keine Systemklassen überschrieben werden.
- Der Sicherheitsmanager (engl. *security manager*) regelt den Zugriff auf Systemressourcen (z.B. Dateisystem, Netzwerk) und verhindert unerlaubte Zugriffe.

Klassenlader und Sicherheitsmanager sind Bestandteil des java.security-Package. Dort findet sich mit den Signaturen (engl. *signatures*) ein weiterer Beitrag zur Sicherheit von Java-Programmen. Dabei wird für den Bytecode mittels einer Einweg-Hashfunktion ein Hashwert berechnet. Dieser wird anschließend nach dem Public-Key-Verfahren (vgl. [Pfleeger 97]) mit dem *private key* des Autors verschlüsselt und an den Bytecode angehängt. Der Benutzer des Bytecodes kann nun den verschlüsselten Hashwert mit Hilfe des *public keys* des Autors wieder entschlüsseln und einen eigenen Hashwert für den Bytecode berechnen. Stimmen die beiden Hashwerte überein, ist der Bytecode authentisch. Es wurde verifiziert, daß er vom angegebenen Autor stammt und nicht manipuliert wurde. Voraussetzung für die Tragfähigkeit dieses Ansatzes ist, daß der Benutzer des Bytecodes die Möglichkeit hat, auf den echten *public key* des Autors zuzugreifen.

Im Kontext dieser Arbeit ist vor allem der Sicherheitsmanager interessant, da er technische Zugriffskontrollen ermöglicht. Im folgenden wird daher ein Überblick über die Funktionsweise des Sicherheitsmanagers gegeben. Detaillierte Informationen finden sich in [Sun 00b].

4.8.2. Der Java-Sicherheitsmanager

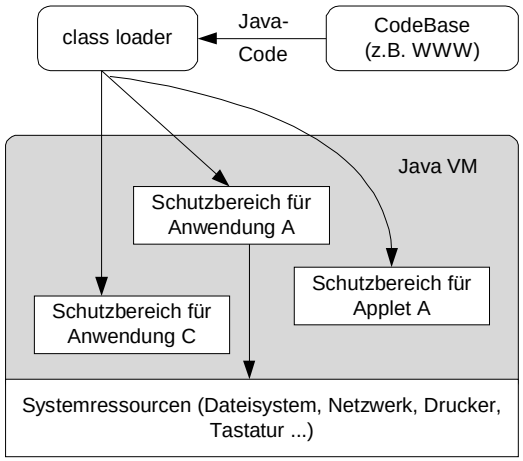


Abbildung 35: Schutzbereiche im Java Sicherheitsmodell (nach [Sun 00b])

Seit der JRE-Version 1.2 ist es möglich, mit Hilfe des Sicherheitsmanagers differenzierte Schutzbereiche (engl. *protection domain*) einzurichten. Ein Schutzbereich wird durch eine Menge von Rechten definiert. Jeder geladenen Klasse kann – entsprechend ihrem Ursprungsort (*CodeBase*) und ihrem Autor – ein solcher Schutzbereich zugewiesen werden (siehe Abbildung 35).

Die Authentizität des Codes wird durch Signieren des Codes sichergestellt. Bei einer potentiell gefährlichen Operation (z.B. Zugriff auf das Dateisystem) überprüft der Sicherheitsmanager, ob diese Operation in dem Schutzbereich ausgeführt werden darf und löst gegebenenfalls eine *SecurityException* aus.

4.8.3. Festlegen von Rechten

Eine Sammlung von Rechten wird *Policy* genannt. Die Rechte, die für eine CodeBase gelten sollen, werden in `.policy`- Dateien gespeichert. Eine solche Datei kann mehrere sogenannte *grant*- Einträge enthalten, mit denen Rechtemengen definiert werden. Eine Rechtemenge kann auf eine CodeBase und/oder einen Code-Autor beschränkt werden.
Im folgenden Beispiel nach [Sun 00b] wird allen Java-Programmen Leserecht im Verzeichnis /tmp/* gegeben, die von der Codebase http://java.sun.com heruntergeladen werden und vom Autor „Li“ signiert sind:

```
grant codeBase "http://java.sun.com" signedBy "Li"
{
  permission java.io.FilePermission "/tmp/*", "read";
};
```

Die `.policy`-Dateien werden beim Start der VM gelesen und zur Laufzeit durch Exemplare der Klasse *java.security.Policy* repräsentiert. Zum Standardumfang des JRE gehören auch diverse Berechtigungen, die als Unterklassen der abstrakten Klasse `java.security.Permission` realisiert sind und den Zugriff auf die Systemressourcen regeln. Diese Klassen bilden direkt die Rechte ab, die in der `.policy`-Datei aufgelistet werden (so gibt es z.B. eine Klasse `java.io.FilePermission`). Selbstdefinierte Rechte können entsprechend durch einfache Unterklassen-Bildung von Permission zum Rechtemodell hinzugefügt und durch den `SecurityManager` überprüft werden. Zur Laufzeit kann ein Java-Programm durch Aufruf von `SecurityManager.checkPermission(...)` überprüfen, ob eine benötigte Berechtigung vorhanden ist.

4.8.4. Beurteilung des Modells

Ziel des Sicherheitsmodells des JRE 1.3 ist es, das sichere Ablaufen von Java-Anwendungen (insbesondere aus fremder Herkunft) auf einem Arbeitsplatzrechner zu gewährleisten. In Bezug auf Zugriffskontrollen geht es ausschließlich um die Absicherung von Zugriffen auf technische Ressourcen, und damit im Sinne von Definition 16 um technische Zugriffskontrollen. Prinzipiell ließe sich das Java-Zugriffskontrollmodell auch für fachliche Zugriffskontrollen verwenden. So könnte man zum Beispiel mit `CustomerListPermission` eine Erlaubnis zum Lesen von einer Kundenliste einführen. `CustomerListPermission` wäre dann eine Unterklasse von `Permission` (vgl. Abbildung 36).

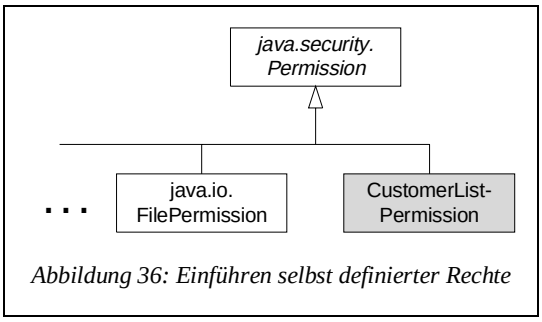


Abbildung 36: Einführen selbst definierter Rechte

Die Definition benutzerbezogener Zugriffsrechte ist allerdings sehr schwierig zu verwirklichen. Im Java-Zugriffskontrollmodell gibt es kein Modell des Benutzers. Lediglich die CodeBase kann verwendet werden, um Zugriffe individuell zuzulassen oder zu verbieten. Benutzerbezogene Zugriffsrechte ließen sich durch eine Zuordnung der CodeBase zu einem Benutzer realisieren. Ein Codebase-Eintrag enthielte dann die aktuelle URL des Arbeitsplatzrechners des Anwenders. Hierzu müßten die CodeBase-Einträge der Policies jedesmal aktualisiert werden, wenn sich ein Benutzer an- oder abmeldet, was technisch sehr aufwendig wäre.

Damit läßt sich zusammenfassend sagen, daß sich fachliche Zugriffskontrollen mit Hilfe des Java-Zugriffskontrollmodells zwar konzeptionell durchführen lassen, eine technische Umsetzung mit adäquatem Aufwand jedoch nicht realisierbar ist.

> **Ergebnis 31**
> Das Java-Zugriffskontrollmodell ist für fachliche Zugriffskontrollen in WAM-Systemen nicht geeignet.

In [Sun 00b] wird darauf hingewiesen, daß es in einer zukünftigen JRE-Version möglicherweise ein explizites Benutzermodell geben wird. Sobald dieses vorliegt, muß das Java-Sicherheitsmodell vor dem Hintergrund seiner Verwendung im WAM/JWAM-Kontext neu bewertet werden.

4.8.5. Verwendung für technische Zugriffskontrollen

Unbeschadet davon, ob das Java-Zugriffskontrollmodell für fachliche Zugriffskontrollen eingesetzt wird, läßt es sich gut für technische Zugriffskontrollen verwenden. Abgesehen vom Schutz technischer Komponenten, die außerhalb eines WAM-Systems existieren (z.B. einer Datenbank), könnte man durch Einführung selbst definierter *Permissions* sowie deren Über-

prüfung gewährleisten, daß nur authentisierte Komponenten miteinander arbeiten. So ließe sich z.B. für eine Registratur sicherstellen, daß nur Werkzeuge bestimmter Werkzeugtypen (also bestimmte Klassen) auf ihr arbeiten. Der Einsatz böswillig manipulierter Werkzeuge ließe sich so zumindest behindern.

4.9. Zusammenfassung

In diesem Kapitel wurde beschrieben, wie ein fachliches Benutzermodell und die Schlüssel-Schloß-Metapher in ein softwaretechnisches Modell umgesetzt werden können. Technischer Hintergrund hierfür ist das JWAM-Rahmenwerk, dessen grundsätzliche Konzepte hier ebenfalls vorgestellt wurden.

Zur Absicherung von Gegenständen wurde das Konzept „verschließbarer Gegenstand“ vorgestellt, welches die Integration von Schlössern in Behälter und Werkzeuge auf einfache Weise ermöglicht. Offen bleibt hingegen, wie Automaten und fachlicher Services auf Entwurfsebene abgesichert werden können.

Komponenten, die verschließbare Gegenstände verwenden, müssen berücksichtigen, daß die Gegenstände abgeschlossen sein können. Dies hat Auswirkungen auf die Konstruktion der Umgebung (beim Starten von einfachen und Kontext-Werkzeugen) sowie auf die Werkzeugkonstruktion.

Bei der Umsetzung der Metapher und des fachlichen Benutzermodells sowie ihrer Integration in andere Entwurfskonzepte ergibt sich, daß in das softwaretechnische Modell technische Anforderungen einfließen. Diese können durch die Metapher auf fachlicher Ebene nicht beschrieben werden. Dies betrifft insbesondere die Absicherung von Werkzeugen. Durch die technisch motivierte Einführung abschließbarer Subwerkzeuge weicht die Handhabung von Softwarewerkzeugen von der Handhabung von Werkzeugen der Anwendungswelt ab.

Bei der Betrachtung des Java-Zugriffskontrollmodells wurde festgestellt, daß es sich aufgrund des Fehlens eines expliziten Benutzermodells nicht für fachliche Zugriffskontrollen für WAM-Systeme eignet. Sofern zur Konstruktion des Anwendungssystems die Programmiersprache Java – und insbesondere das JWAM-Rahmenwerk – eingesetzt wird, kann das Java-Sicherheitsmodell für technische Zugriffskontrollen verwendet werden.

Durch die Einführung benutzerbezogener Zugriffskontrollen ergeben sich Konsequenzen für den Arbeitsplatz und den Desktop: der Arbeitsplatz eines Anwenders soll abschließbar sein und abgeschlossene Gegenstände sollen auf dem Desktop besonders gekennzeichnet werden. Auf diese Weise bemerkt der Anwender, daß er sie nicht benutzen kann. Diese Anforderungen konnten im Rahmen dieser Arbeit nicht berücksichtigt werden, da sich ihre Umsetzung als zu aufwendig erwies. Dies wird in zwei offenen Fragen festgehalten:

Offene Frage 10
Wie kann das Abschließen des persönlichen Arbeitsplatzes konstruktiv umgesetzt werden?

Offene Frage 11
Wie lassen sich abgeschlossene Gegenstände auf dem Desktop besonders kennzeichnen?

Kapitel 5 - Konstruktion einer Zugriffskontrolle für das JWAM-Rahmenwerk

Wie im vorangegangenen Kapitel bereits angedeutet wurde, ist die Schlüssel-Schloß-Metapher im Rahmen dieser Arbeit konstruktiv umgesetzt und in das JWAM-Rahmenwerk integriert worden. Ausgangspunkt der Konstruktion ist der ebenfalls in Kapitel 4 vorgestellte fachliche Entwurf. Der Quellcode der Implementation aller neu eingeführten Klassen findet sich im Package de.jwam.handling.accesscontrol. Das Rahmenwerk selbst kann über www.jwam.de heruntergeladen und evaluiert werden[22].

Aufgrund terminlicher Absprachen mit den Mitgliedern der JWAM-Architekturgruppe war es nicht möglich, den hier vorgestellten aktuellen Stand der Konstruktion vor Abgabe der Arbeit in das Rahmenwerk zu integrieren. Dies betrifft insbesondere die in Abschnitt 5.4 vorgestellte Schnittstellen-Hierarchie sowie die Konstruktion hierarchischer Schlüssel (Abschnitte 5.2.1, 5.2.2), welche für die derzeit aktuelle Rahmenwerks-Version 1.5.1 nicht mehr überarbeitet werden konnten.

5.1. Modellierung von Benutzern

Die wesentlichen Elemente dieses Entwurfes sind das Benutzermodell selbst, ein Fachwert als fachlicher Verweis und eine einfache Benutzerorganisation (s. Abschnitt 4.2.4). Diese werden in der Konstruktion durch die Klassen `User`, `dvUserIdentificator` und `UserOrganisation` modelliert. Der Schlüsselbund (`JwamKeyRing`, siehe Abschnitt 5.2) beinhaltet die Schlüssel eines Anwenders während seine Zertifikate in einem `ContainerImpl`-Exemplar aufbewahrt werden. Einen Überblick über die Konstruktion und ihre Einbettung in das Rahmenwerk bietet Abbildung 37. Dabei sind hier und in den folgenden Klassendiagrammen alle Klassen, die im Rahmen dieser Arbeit zum Rahmenwerk hinzugefügt wurden, grau unterlegt.

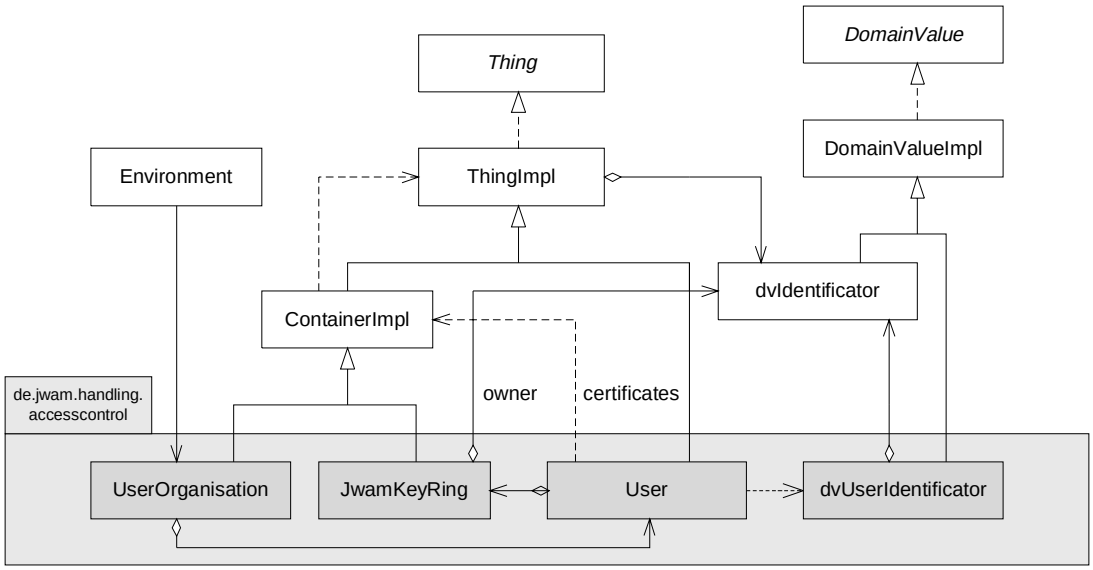


Abbildung 37: Konstruktion von User und UserOrganisation

5.1.1. dvUserIdentificator

Bevor auf die eigentliche Konstruktion von `User` eingegangen wird, soll der Fachwert `dvUserIdentificator` vorgestellt werden. Dieser Fachwert spielt in Bezug auf die Zu-

[22] Das JWAM-Rahmenwerk wird durch die APCON WPS (Hamburg) in enger Zusammenarbeit mit dem Arbeitsbereich Softwaretechnik des Fachbereichs Informatik der Universität Hamburg entwickelt. Die Rechte für die kommerzielle Nutzung des Rahmenwerks liegen bei APCON WPS.

griffskontrolle selbst eine geringe Rolle, ist allerdings für andere Komponenten von großer Bedeutung. Beispiele sind die Registratur und das Postversandsystem (siehe Abschnitt 3.7.2), wo `dvUserIdentificator` als fachlicher Verweis Verwendung findet wird. Wird ein Gegenstand hineingelegt oder herausgenommen wird, so besagt der hinterlegte `dvUserIdentificator`-Wert, welcher Anwender diese Aktion veranlaßt hat.

Die reine Benutzeridentifizierung hätte, wie bei vielen anderen Gegenständen auch, durch ein `dvIdentificator` geschehen können. Darüber hinaus soll `dvUserIdentificator` auch zur Repräsentation des Benutzers dienen können. Zu diesem Zweck beinhaltet er eine Benutzerkennung in Form des Login-Namens des Benutzers.

Eine weitere Motivation zur Einführung `dvUserIdentificator` erwächst aus zwei technisch bedingten Schwächen von `dvIdentificator`. Wenn statt technischer Referenzen `dvIdentificator`-Werte benutzt werden, besteht die Schnittstelle vieler Operationen aus mehreren Parametern vom Typ `dvIdentificator`. Eine solche Schnittstelle wird unübersichtlich, da nur noch am Namen des Parameters seine Funktionalität erkennbar ist. Zudem läßt sich für `dvIdentificator` kein gültiger Standardwert erzeugen, wie er zur Repräsentation des Standardbenutzers benötigt wird (vgl. Abschnitt 4.2.3).

Als Lösung wurde `dvUserIdentificator` so entworfen, daß er den Login-Namen des Benutzers sowie einen `dvIdentificator` umhüllt. Damit kann er zur Repräsentation und Identifikation dienen und den erforderlichen Standardwert erzeugen. Daneben gibt es auch einen undefinierten Wert. Dieser Wert soll zum Ausdruck bringen, daß es keinen gültigen Benutzer-Wert als Ergebnis einer Operation gibt.

```
public class dvUserIdentificator extends DomainValueImpl
{
   public dvIdentificator id();
   public String name();
   public boolean isDefaultUser();
   public static class Factory extends DomainValueImpl.Factory
   {
      public DomainValue undefinedValue();
      public DomainValue defaultValue ();
      public static DomainValueImpl valueOf (dvIdentificator id, String name);
      public DomainValueImpl value (dvIdentificator id, String name);
   }
}
```

Code 2: Ausschnitt aus der Schnittstelle von dvUserIdentificator

Der in Code 2 dargestellte Schnittstellen-Ausschnitt[23] verdeutlicht auch das Prinzip der Konstruktion von Fachwerten. Fachwerte bestehen aus den fachlichen Operationen (in diesem Fall `id(), name()`und `isDefaultUser()`) sowie einer Fabrik, die die Fachwerte erzeugt.

Die Fabrik ist in der Oberklasse `DomainValueImpl` nach dem *Singleton*-Entwurfsmuster[24] implementiert und für die Erzeugung von Fachwert-Exemplaren zuständig. Ein `dvUserIdentificator` für einen konkreten Benutzer läßt sich mit Hilfe von `value(...)` bzw. `valueOf(...)` erzeugen. Die Operationen unterscheiden sich lediglich dadurch, daß `valueOf(...) static` ist und damit nicht erst die `instance()` Methode der Oberklasse gerufen werden muß, um an die Fabrik zu gelangen. Der Konstruktor ist wie bei allen Fachwerten `protected`, denn die Erzeugung von Fachwerten soll nur über die zuständige Fabrik geschehen. Die Parameter von `value(...)` bzw. `valueOf(...)` sind der `dvIdentificator` des korrespondierenden User-Exemplars sowie dessen Login-Name.

[23] Hier und im folgenden werden Klassenschnittstellen in dem Rahmen präsentiert, der zum Verständnis der Konstruktion erforderlich ist. Hierzu gehören auch Vor- und Nachbedingungen.

[24] Eine Klasse, die nach dem *Singleton*-Entwurfsmuster implementiert ist, besitzt in einem Prozeßraum genau ein Exemplar. Dieses wird über einen global erreichbaren Zugriffspunkt bereitgestellt. Näheres siehe [Gamma et al. 96] (Seite 157ff).

5.1.2. Konstruktion: User

Die Klasse `User` ist eine Unterklasse von `ThingImpl`. Konstruiert werden `User`-Objekte ausschließlich von der Benutzerorganisation (`UserOrganisation`, s.u.). Der Konstruktor von `User` ist dementsprechend `protected`. Einen Ausschnitt der Schnittstelle zeigt Code 3.

```
public class User extends ThingImpl
{
   public boolean isPasswordValid (String password);
   // @require isPasswordValid(oldPassword)
   public void changePassword(String oldPassword, String newPassword);
   public dvUserIdentificator userIdentificator ();
   public void addCertificate (JwamKeyCertificate certificate);
   public boolean hasCertificate (dvIdentificator id);
   public dvTableOfContents keyCertificateDescriptions ();
   public boolean isDefaultUser ();
   public KeyRing keyRing ();
   protected JwamKeyCertificate certificate (dvIdentificator id);
   protected void removeCertificate (dvIdentificator id);
}
```

Code 3: Ausschnitt aus der User-Schnittstelle

Während sich die meisten Parameter von `User` einfach ändern lassen, kann das Paßwort des Benutzers durch die Operation `changePassword(...)` nur bei Kenntnis des alten Paßwortes verändert werden.

Die Verwaltung von Zertifikaten obliegt einem internen fachlichen Behälter. Von außen können mit `addCertificate(...)` Zertifikate hinzugefügt sowie mit `hasCertificate(...)` überprüft werden, ob ein `User` über ein bestimmtes Zertifikat verfügt. Die Methode `keyCertificateDescriptions()` gibt einen Fachwert zurück, der ein Inhaltsverzeichnis der beinhalteten Zertifikate repräsentiert (vgl. Abschnitt 4.1.6). Schließlich kann mit `isDefaultUser()` überprüft werden, ob es sich bei dem `User`-Exemplar um den Standard-Benutzer handelt. Mit der Operation `keyRing()` erhält man den Schlüsselbund des Benutzers. Mit `certificate(...)` wird eine Referenz auf ein Zertifikat des Benutzers herausgegeben und mit `removeCertificate(...)` kann es gelöscht werden. Diese beiden Operationen sind `protected`, da von außerhalb des Package keine unbefugten Manipulationen an Zertifikaten vorgenommen werden sollen.

5.1.3. Konstruktion einer einfachen Benutzerorganisation

In Abschnitt 4.2.4 wurde vorgeschlagen, die Benutzerorganisation als fachlichen Service zu modellieren. Zum Zeitpunkt der Verfassung dieser Arbeit war im JWAM-Rahmenwerk nur eine rudimentäre Unterstützung zur Konstruktion fachlicher Services vorhanden. Da der Schwerpunkt der Konstruktion auf der Umsetzung der Schlüssel-Schloß-Metapher lag, wurde eine stark vereinfachte Version einer Benutzerorganisation konstruiert. Diese weist konzeptionell Ähnlichkeiten mit einem fachlichen Behälter auf, was sich im Rahmen dieser Arbeit als ausreichend erwies. Als offene Frage verbleibt daher:

> **Offene Frage 12**
> Wie kann die Benutzerorganisation als fachlicher Service konstruiert werden?

Die Benutzerorganisation wird durch die Klasse `UserOrganisation` modelliert, welche eine Unterklasse von `ContainerImpl` ist. `ContainerImpl` ist die Basisklasse zur Konstruktion fachlicher Behälter im Rahmenwerk (vgl. Abschnitt 4.1.6). Einen Überblick über die Schnittstelle von `ContainerImpl` verschafft Code 4. Diese Schnittstelle arbeitet im wesentlichen auf Ebene von `Thing` bzw. `dvIdentificator`, so daß das Hinzufügen und Herausnehmen von Gegenständen (durch Aufruf von `add(...)` bzw. `remove(...)`) einheitlich auf Ebene der Oberklasse geregelt wird. Mit `isRemovable(...)` und `isAddable(...)` kann geprüft werden, ob ein Gegenstand entfernt oder hineingelegt werden darf. Diese Ope-

rationen werden oft in einer Unterklasse überschrieben, wenn das Hinzufügen bzw. Herausnehmen von Gegenständen in speziellen Behältern an weitere Bedingungen geknüpft ist.

```
public class ContainerImpl extends ThingImpl implements Container
{
   public boolean isRemovable (dvIdentificator id);
   // @require isRemovable(id)
   public void remove (dvIdentificator id);
   public int count ();
   public boolean has (dvIdentificator id);
   public dvTableOfContents tableOfContents();
   public boolean isAddable (Thing thing);
   // @require isAddable (thing)
   public void add (Thing thing);
}
```

Code 4: Ausschnitt aus der Schnittstelle von ContainerImpl

Code 5 zeigt einen Ausschnitt aus der Schnittstelle von `User`. `User`-Exemplare werden durch die `newUser(...)` Operation erzeugt, wobei die Parameter denen im Konstruktor von `User` weitgehend entsprechen. Einzige Bedingung bei der Erzeugung eines Benutzers ist die Eindeutigkeit seines Login-Namens. Mit `has(userLoginName)` kann vor Aufruf von `newUser(...)` überprüft werden, ob bereits ein Benutzer mit dem entsprechenden Login-Namen registriert ist. Die Operation `defaultUser()` erzeugt das `User`-Exemplar, welches den Standard-Benutzer repräsentiert.

Die restlichen Operationen ergänzen die `ContainerImpl`-Schnittstelle (s.o.). Eine gesonderte Anpassung der Schnittstelle an `dvUserIdentificator` ist nicht nötig, da `dvUserIdentificator` denselben `dvIdentificator` Wert wie `User` beinhaltet, der einfach über `dvUserIdentificator.id()` erfragt werden kann. Die Implementation von `isAddable(...)` stellt sicher, daß dem Behälter nur `User`-Objekte hinzugefügt werden.

```
public class UserOrganisation extends ContainerImpl
{
   public UserOrganisation ();
   public boolean isAddable (Thing thing);
   public User defaultUser ();
   // @require !has(id) && !has(loginName)
   public User newUser (...);
   public boolean has (String userLoginName);
   public dvIdentificator userId (String userLoginName);
   public User user (dvIdentificator id);
   public User user (String userLoginName);
   public void save();
}
```

Code 5: Ausschnitt aus der UserOrganisation-Schnittstelle

Gespeichert wird die `UserOrganisation` bei Aufruf der Methode `save()` durch die Umgebung. Eine Standardimplementation sieht die Speicherung im Dateisystem vor.

5.2. Modellierung von Schlüsseln, Schlössern und Zertifikaten

Der Zusammenhang zwischen Schlüsseln, Schlössern und Zertifikaten ist in Kapitel 3 ausführlich dargestellt worden. In diesem Abschnitt wird die Implementierung für das JWAM-Rahmenwerk vorgestellt. Einen Überblick über die Konstruktion verschafft Abbildung 38.

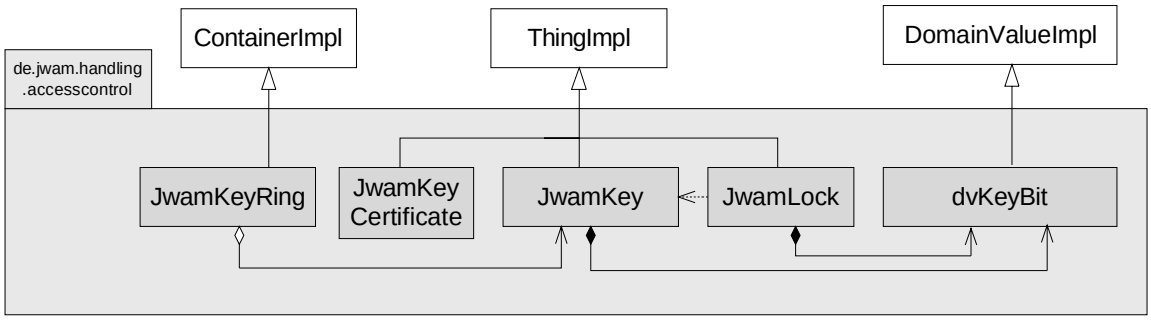


Abbildung 38: Überblick über die Konstruktion von Schlüsseln, Schlössern und Zertifikaten

`JwamKey`[25] und `JwamLock` modellieren einen Schlüssel bzw. ein Schloß. Bei `JwamKeyCertificate` handelt es sich um das Zertifikat und bei `dvKeyBit` um den Fachwert, der den Schlüsselbart repräsentiert.

5.2.1. Der Schlüsselbart

Ein Schlüsselbart wird durch die Klasse `dvKeyBit` modelliert (siehe Code 6). Die interne Fabrikklasse (vgl. Abschnitt 5.1.1) wird nicht dargestellt, da ihre Schnittstelle keine besondere Veränderung gegenüber der Basisklasse aufweist.

Im Konstruktor werden der `dvIdentificator` des zugehörigen Zertifikates, sowie gegebenenfalls die `dvKeyBits` aller Sub-Schlüssel übergeben. Der `dvIdentificator` des Zertifikates bildet den Sicherheitscode ab, der auf einen Sicherheitsschlüssel geprägt ist. Abgefragt wird dieser Wert mit `certificateId()`.

```
public class dvKeyBit extends DomainValueImpl
{
   protected dvKeyBit (dvIdentificator certificateId, dvKeyBit[] superKeyBits);
   public dvIdentificator certificateId ();
   public dvIdentificator[] directSuperKeybitCertficateIds ();
   public boolean equals (Object bit);
   protected boolean isSubKeyBitOf (dvKeyBit bit);
   protected boolean isSuperKeyBitOf (dvKeyBit bit);
}
```

Code 6: Ausschnitt aus der Schnittstelle von dvKeyBit

Als Merkmal enthält `dvKeyBit` ebenfalls einen `dvIdentificator`. Dieser `dvIdentificator` hat nichts mit dem des Zertifikates zu tun. Während der `dvIdentificator` des Zertifikates außerhalb der Klasse bekannt sein soll, wird der interne `dvIdentificator` von `dvKeyBit` geheimgehalten, damit er nicht manipuliert oder kopiert wird.

Der Bart eines Sub-Schlüssels beinhaltet die Merkmale seiner Super-Schlüssel. In der Konstruktion wird dies durch eine interne Baumstruktur der Schlüsselbärte realisiert: ein hierarchisch untergeordnetes `dvKeyBit` kennt alle direkten Super- `dvKeyBits`. Dieser Ansatz erlaubt es, später die hierarchische Anordnung zu rekonstruieren. Nur aufgrund der hierarchischen Anordnung läßt sich beim Auf- oder Abschließen eines Schlosses entscheiden, ob zwei Schlüssel in einer Sub-Schlüssel / Super-Schlüssel Beziehung zueinander stehen. Hierzu dienen die Operationen `isSubKeyBitOf(...)` und `isSuperKeyBitOf(...)`. Mit `directSuperKeybitCertficateIds()` können die Zertifikate der direkten Super-Schlüssel abgefragt werden. Zwei `dvKeyBit`-Exemplare sind gleich, wenn sie dasselbe Merkmal enthalten und dieselben Super-Schlüsselbärte beinhalten.

[25] Der Präfix „Jwam“ wird gewählt, damit es nicht zu Mißverständnissen durch die Begriffsverwendung kommt. Eine Schnittstelle „Key“ ist Bestandteil des java.security Package. Allerdings geht es dort um kryptographische Schlüssel, ein Bezug zu Zugriffskontrollen ist nicht gegeben.

5.2.2. Schlüssel

Schlüssel werden im JWAM-Rahmenwerk durch die Klasse `JwamKey` modelliert (siehe Code 7). `JwamKey` ist eine Unterklasse von `ThingImpl`. Hierdurch wird die Gegenständlichkeit von Schlüsseln zum Ausdruck gebracht. Der Aufbau von `JwamKey` ist sehr einfach, denn `JwamKey` stellt nicht mehr als eine Hülle um `dvKeyBit` dar. Diese Umhüllung ist notwendig, da `dvKeyBit` kein Gegenstand ist. Daher werden die Operationsaufrufe `isSubKeyOf(...)`, `isSuperKeyOf(...)` und `certificateId(...)` an `dvKeyBit` weitergeleitet. Die Operation `thingDescription()` überschreibt die Operation der Oberklasse, um eine konkrete Repräsentation für Schlüssel festzulegen. Da `JwamKeys` ausschließlich von der `KeyLockFactory` (s.u.) erzeugt werden, ist der Konstruktor `protected`. Das `dvKeyBit` für diesen Schlüssel wird im Konstruktor übergeben und kann danach nicht mehr ausgetauscht werden.

```
public class JwamKey extends ThingImpl
{
   protected JwamKey (String keyName, dvKeyBit keyBit);
   public dvIdentificator certificateId ();
   public boolean isSubKeyOf (JwamKey key);
   public boolean isSuperKeyOf (JwamKey key);
   public dvThingDescription thingDescription ();
   protected dvKeyBit keyBit ();
}
```

Code 7: Ausschnitt aus der Schnittstelle von JwamKey

Es fällt auf, daß keine Operationen für die Zusammenarbeit mit dem Schloß definiert sind. Hierauf wird im folgenden Abschnitt bei der Diskussion von Schlössern eingegangen.

5.2.3. Schlüsselbund

Der Schlüsselbund wird durch die Klasse `JwamKeyRing` modelliert (siehe Code 8). Wie an einen echten Schlüsselbund können Schlüssel hinzugefügt und wieder entfernt werden. Diese und andere übliche Behälter-Operationen sind durch die Oberklasse `ContainerImpl` definiert (siehe auch Abschnitt 5.1.3). `JwamKeyRing` reimplementiert `isAddable()`, so daß nicht alle `Things`, sondern nur `JwamKeys` als Parameter der `add()` Operation akzeptiert werden. Zusätzlich hat ein Schlüsselbund einen Besitzer (hier durch `dvUserIdentificator` repräsentiert). Der Besitzer des Schlüsselbundes ist fachlich ebenfalls der Besitzer der Schlüssel am Schlüsselbund.

```
public class JwamKeyRing extends ContainerImpl
{
   public KeyRing (String name, dvUserIdentificator owner);
   public dvUserIdentificator owner ();
   public JwamKey key(dvIdentificator id);
   public boolean isAddable (Thing thing);
   public JwamKey[] keys ();
}
```

Code 8: Ausschnitt aus der Schnittstelle von JwamKeyRing

5.2.4. Schlösser

Schlösser werden im JWAM-Rahmenwerk durch die Klasse `JwamLock` modelliert (siehe Code 9). Auch `JwamLock` erbt von `ThingImpl`, und Exemplare werden durch die `KeyLockFactory` erzeugt. Mit `isLocked()` läßt sich überprüfen, ob ein Schloß verschlossen ist und `fitsIn(...)` ermittelt, ob ein bestimmter Schlüssel in das Schloß paßt. Nur in diesem Fall kann es durch die Operationen `unlock(...)` bzw. `lock(...)` ab- oder aufgeschlossen werden. Die Operation `fitsIn(keyRing)` ergibt `true`, wenn mindestens ein Schlüssel des Schlüsselbundes in das Schloß paßt.

```
public class JwamLock extends ThingImpl
{
   protected JwamLock (String name, dvKeyBit keyBit, boolean isLocked);
   public boolean fitsIn (JwamKey key);
   public boolean fitsIn (KeyRing keyRing);
   // @require fitsIn(key)
   public void unlock (JwamKey key);
   // @require fitsIn(keyRing)
   public void unlock (KeyRing keyRing);
   // @require fitsIn(key)
   public void lock (JwamKey key);
   // @require fitsIn(keyRing)
   public void lock (KeyRing keyRing);
   public boolean isLocked();
   public dvIdentificator certificateId();
}
```

Code 9: Ausschnitt aus der Schnittstelle von JwamLock

`JwamLock` verfügt wie `JwamKey` über ein `dvKeyBit`. Wenn ein Schlüssel ein Schloß aufschließen soll, wird überprüft, ob die `dvKeyBits` von Schloß und Schlüssel gleich sind oder das `dvKeyBit` des Schlosses ein Sub-`dvKeyBit` des Schlüssels ist. Ist dies der Fall, wird das Schloß geöffnet. Das zugehörige Zertifikat kann wie beim Schlüssel durch Aufruf von `certificateId()` ermittelt werden, wobei auch hier der Operationsaufruf an das `dvKeyBit` weitergeleitet wird.

Zusammenspiel zwischen Schlüssel und Schloß

Nachdem die Klassen `JwamKey`, `JwamLock` und `dvKeyBit` beschrieben wurden, kann ihr Zusammenspiel erläutert werden. Dies geschieht am Beispiel eines abgeschlossenen Behälters, dessen Schloß (z.B. von einem Werkzeug) geöffnet werden soll (siehe Abbildung 39).

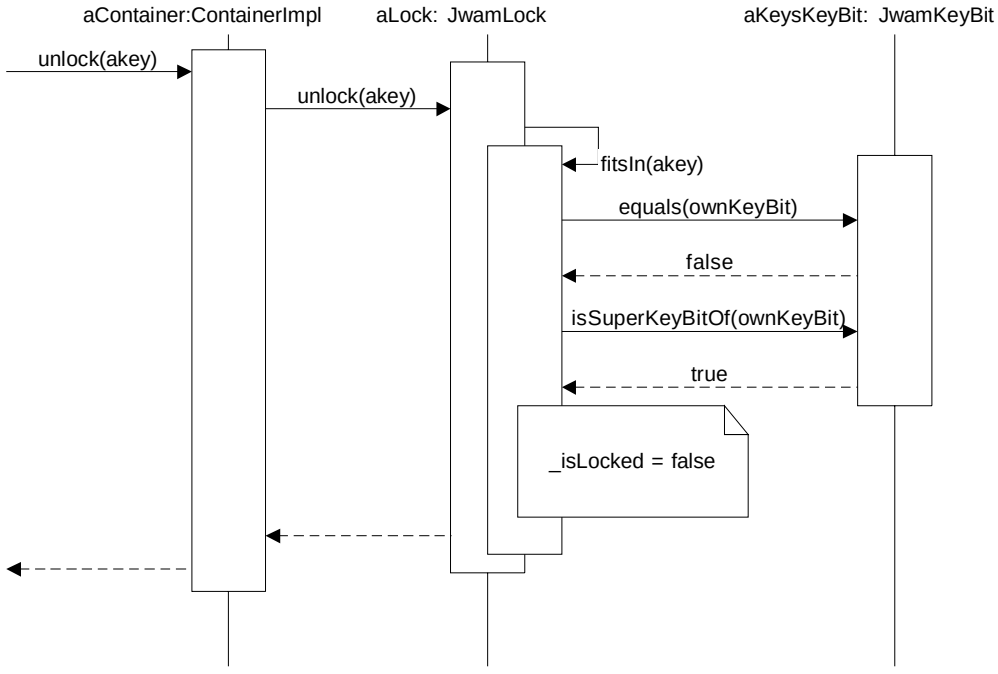


Abbildung 39: Erfolgreiches Aufschließen eines Schlosses

Der Behälter verfügt als verschließbarer Gegenstand über ein Schloß. Nach dem Aufruf von `unlock(akey)` überprüft das Schloß, ob der übergebene Schlüssel paßt. Hierzu wird das `dvKeyBit` des Schlosses mit dem des Schlüssels verglichen. Die `dvKeyBits` sind nicht identisch, allerdings ist das `dvKeyBit` des Schlüssels dem des Schlosses hierarchisch übergeordnet (`isSuperKeyBitOf()` ergibt `true`). Anschließend wird der Zustand des Schlosses

verändert, und das Schloß ist offen. Das Schloß sondiert den Schlüssel und entscheidet aufgrund der Beschaffenheit des `dvKeyBits`, daß es geöffnet werden kann. Auf die Darstellung von Vor- und Nachbedingungen wird in diesem Beispiel aus Gründen der Übersichtlichkeit verzichtet.

5.2.5. Zertifikate

Ein Zertifikat wird durch die Klasse `JwamKeyCertificate` modelliert (siehe Code 10), erbt von `ThingImpl` und wird wie Schlüssel und Schlösser durch die `KeyLockFactory` erzeugt.

```
public class JwamKeyCertificate extends ThingImpl
{
   protected JwamKeyCertificate (dvIdentificator vendor, String
          name, dvIdentificator keyBitId, ...);
   public dvIdentificator vendor ();
   public dvIdentificator keyBitId ();
}
```

Code 10: Ausschnitt aus der Schnittstelle von JwamCertificate

Wenn ein Exemplar von `JwamKeyCertificate` erzeugt wird, erhält es als Parameter unter anderem einen `dvIdentificator` (bezeichnet mit `vendor`), der auf die `KeyLockFactory` verweist, die dieses Zertifikat herausgegeben hat. Dies spielt eine Rolle, sobald es in einem Anwendungssystem mehrere Fabriken gibt.

Bei der Erzeugung eines Zertifikates wird auch ein `dvKeyBit` generiert. Das `JwamKeyCertificate` verweist mit Hilfe eines `dvIdentificator` auf dieses zugehörige `dvKeyBit`. Sein `dvIdentificator` kann mittels `keyBitId()` abgefragt werden.

5.3. Erzeugung von Schlüsseln und Schlössern

Im vorangegangen Abschnitt wurde mehrfach erwähnt, daß die Klassen `JwamKeyCertificate`, `JwamKey` und `JwamLock` sowie `dvKeyBit` von einer weiteren Klasse namens `KeyLockFactory` erzeugt werden. Diese Klasse hat den Charakter einer Fabrik, implementiert allerdings nicht eines der in [Gamma et al. 96] vorgestellten Fabrikmuster. Diese Art der Erzeugung wurde gewählt, da die vier genannten Klassen sowohl fachlich als auch technisch sehr eng zusammenhängen. Dieser Zusammenhang kann sinnvoll nur durch eine erzeugende Klasse, die `KeyLockFactory`, gewährleistet werden.

Konzeptionell gesehen handelt es sich bei der `KeyLockFactory` um einen fachlichen Service. Analog zur Benutzerorganisation wurde die Implementation vereinfacht. Die `KeyLockFactory` wurde nach dem Singleton-Entwurfsmuster für eine lokale Arbeitsplatzumgebung realisiert. Wie die `KeyLockFactory` als fachlicher Service modelliert werden kann, konnte im Rahmen dieser Arbeit nicht untersucht werden.

> **Offene Frage 13**
> Wie kann die `KeyLockFactory` als fachlicher Service konstruiert werden?

Neben der reinen Erzeugung der genannten Klassen erfüllt die `KeyLockFactory` auch eine wichtige Funktion als Archiv für erzeugte Exemplare. Damit im nachhinein nachvollziehbar ist, für welchen Benutzer welche Schlüssel, Schlösser und Zertifikate erzeugt wurden, werden die entsprechenden Informationen durch die Fabrik archiviert. Dies kann zum Beispiel wichtig sein, wenn ein Mitarbeiter ein Unternehmen verläßt und rekonstruiert werden soll, welche `JwamKey`-Exemplare ihm persönlich ausgehändigt wurden. Einen Überblick über die Konstruktion verschafft Abbildung 40.

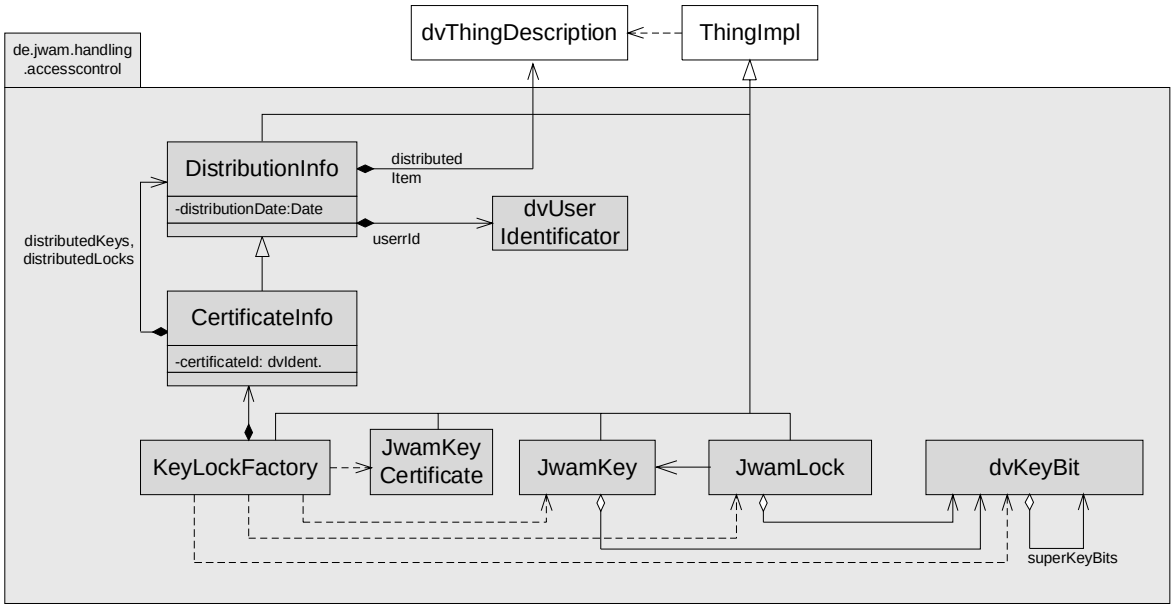


Abbildung 40: Konstruktion von Schlüsseln, Schlössern und Zertifikaten im Detail

5.3.1. Konstruktion: Die Fabrik KeyLockFactory

`KeyLockFactory` erbt von `ThingImpl`. Ihre Funktion besteht in der Erzeugung und Verwaltung von Schlüsseln, Schlössern und Zertifikaten. Bei der Erzeugung eines Schlüssels, Schlosses oder Zertifikates sind mehrere Arbeitsschritte erforderlich. Diese werden von der Fabrik nach Aufforderung ausgeführt.

```
public class KeyLockFactory extends ThingImpl
{
  protected KeyLockFactory ();
  public static KeyLockFactory instance ();
  public JwamKeyCertificate newCertificate(String name, dvUserIdentificator owner);
  public JwamKeyCertificate subCertificate(String name,
     dvUserIdentificator owner, dvIdentificator[] superCertificateIds);
  public JwamLock newLock (JwamKeyCertificate certificate,
     String name, dvUserIdentificator owner);
  public JwamKey newKey (JwamKeyCertificate certificate, String name,
     dvUserIdentificator owner);
  public dvIdentificator[] certificateIds();
  public boolean has (dvIdentificator certificateId);
  // @require has (certificateId)
  public CertificateInfo certificateInfo (dvIdentificator certificateId);
  // @require has (certificateId)
  public dvUserIdentificator owner (dvIdentificator certificateId);
  public void save();
}
```

Code 11: Ausschnitt aus der Schnittstelle von KeyLockFactory

Code 11 zeigt einen Ausschnitt aus der Schnittstelle von `KeyLockFactory`. Wie bei Singletons üblich, liefert `instance()` eine Referenz auf das Exemplar. Mit `newCertificate(...)` bzw. `subCertificate(...)` lassen sich neue Zertifikate erzeugen, wobei im zweiten Fall ein Zertifikat erzeugt wird, das den übergebenen Zertifikaten hierarchisch untergeordnet ist. Die Methoden `newLock(...)` bzw. `newKey(...)` erzeugen neue Schlösser bzw. Schlüssel. Einige sondierende Funktionen ermöglichen es, näheres zu den erzeugten Zertifikaten zu erfragen: `certificateIds()` liefert die `dvIdentificator` -Werte der bereits erzeugten Zertifikate, `has(...)` überprüft, ob ein Zertifikat mit einem bestimmten `dvIdentificator` von der Fabrik erzeugt wurde und `owner(...)` ermittelt den Anwen-

der, für den das Zertifikat registriert ist. Mit `certificateInfo(...)` können weitere Informationen zu einem Zertifikat ermittelt werden (näheres hierzu in Abschnitt 5.3.4).

Die von der `KeyLockFactory` verwalteten Informationen werden durch Aufruf der Methode `save()` durch die Umgebung gespeichert. Als Standard-Persistenzmedium ist das Dateisystem vorgesehen. Durch entsprechendes Überschreiben von `save()` kann auch ein anderes Persistenzmedium angebunden werden.

5.3.2. Erzeugen eines Zertifikates

Ein neues Zertifikat wird mit `newCertificate(...)` bzw. `subCertificate(...)` erzeugt. Der Anwender bestimmt den Namen des Zertifikates sowie welche Super-Zertifikate gegebenenfalls zu dem neuen Zertifikat gehören. Der Name des Zertifikates sollte Auskunft darüber geben, wofür zugehörige Schlüssel und Schlösser verwendet werden sollen.

Bei der Generierung eines Zertifikates wird auch ein `dvKeyBit` erzeugt und aufbewahrt. Alle später für das Zertifikat erzeugten Schlüssel und Schlösser werden dieses `dvKeyBit` erhalten. Neben dem `dvKeyBit` verwaltet die Fabrik noch weitere Informationen über ihre Zertifikate in einer Klasse `CertificateInfo` (s.u.), welche sie in einem internen Behälter speichert.

Bei der Erzeugung des Zertifikates wird sein Besitzer registriert. Hierzu dient der Parameter `dvUserIdentificator`.

5.3.3. Erzeugen von Schlüsseln und Schlössern

Schlüssel und Schlösser werden durch die Operationen `newKey(...)` bzw. `newLock(...)` erzeugt. Als Parameter muß auf jeden Fall ein Zertifikat mitgegeben werden. Dabei werden ein Name für den Gegenstand sowie sein Besitzer angegeben. Wenn ein Schlüssel oder Schloß erzeugt wird, merkt sich die Fabrik die `dvThingDescription` sowie Besitzer und Zeitpunkt der Erzeugung in der Klasse `DistributionInfo`.

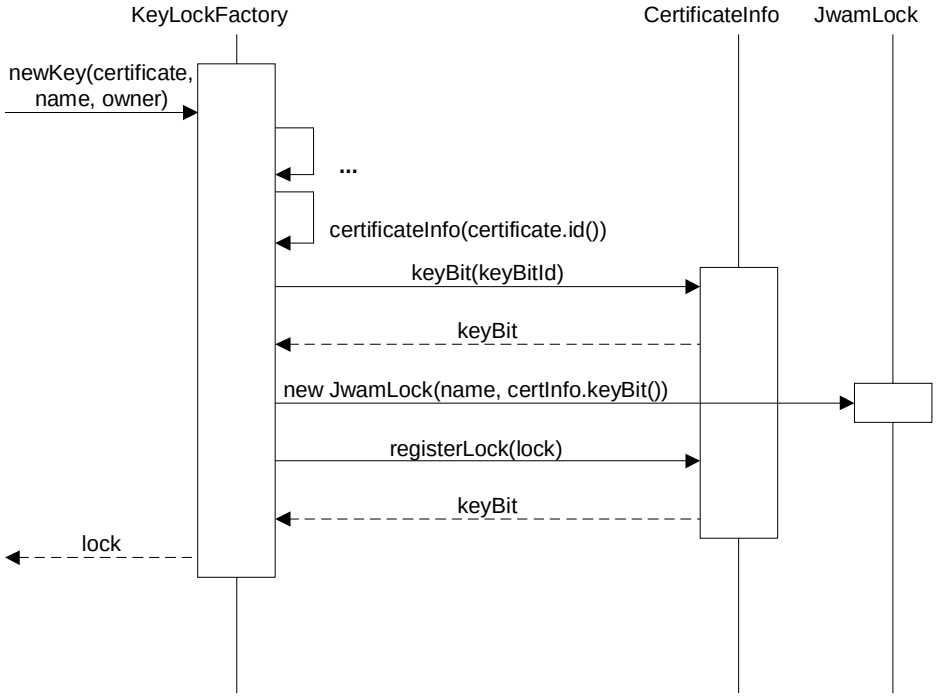


Abbildung 41: Erzeugung eines Schlosses

Der Ablauf der Erzeugung eines Schlosses wird exemplarisch in Abbildung 41 dargestellt. Nach der Aufforderung, ein neues Schloß zu erzeugen (`newLock(...)`), sucht die Fabrik zunächst das `CertificateInfo`-Exemplar (s.u.) heraus, welches zu dem übergebenen Zertifikat gehört. In diesem ist der `dvKeyBit`-Wert gespeichert, welcher für das neue Schloß be-

nötigt wird. Der Wert wird vom `CertificateInfo`-Exemplar abgefragt und in das mit `newLock(...)` erzeugte Schloß eingebaut. Bevor die Fabrik das Schloß herausgibt, trägt sie im `CertificateInfo`-Exemplar ein, daß ein neues Schloß erzeugt wurde. Das `CertificateInfo`-Exemplar speichert dies in einem `DistributionInfo`-Exemplar.

5.3.4. Konstruktion: DistributionInfo und CertificateInfo

Die `KeyLockFactory` registriert Informationen über die von ihr erzeugten Gegenstände. Hierzu dient ein Material `DistributionInfo` (siehe Code 12), dessen Einführung rein technisch motiviert ist. `DistributionInfo` speichert, für welchen Benutzer ein Gegenstand wann erzeugt wurde. `DistributionInfos` benötigen keine Referenz auf das erzeugte `Thing`, statt dessen wird ein Fachwert `dvThingDescription` benutzt. Die entsprechenden Informationen werden im Konstruktor übergeben und können mit `userId()`, `date()` und `thingDescription()` abgefragt werden. Mit `changeUserId(...)` kann ein neuer Benutzer als Besitzer des Gegenstandes registriert werden.

```
public class DistributionInfo extends ThingImpl
{
   protected DistributionInfo (dvThingDescription thingDescription,
      dvUserIdentificator userId, Date date);
   public dvThingDescription description();
   public dvUserIdentificator userId();
   public Date date ();
   protected void changeUserId (dvUserIdentificator userId);
}
```

Code 12: Ausschnitt aus der Schnittstelle von DistributionInfo

Bei der Erzeugung eines Zertifikates müssen weitere Informationen gespeichert werden. Hierzu wird mit `CertificateInfo` (siehe Code 13) eine Unterklasse von `DistributionInfo` gebildet. Ein `CertificateInfo`-Exemplar erhält bei seiner Erzeugung die `dvThingDescription` des Zertifikates. Wenn die Fabrik zum Beispiel ein neues Schloß erzeugt, sucht sie die zum Zertifikat zugehörige `CertificateInfo` und registriert das Schloß dort mit `registerLock(...)`. Daraufhin wird eine DistributionInfo für das Schloß erzeugt und in der `CertificateInfo` gespeichert. Die Fabrik kann die `CertificateInfo` eines `JwamCertificate` herausgeben, was zum Beispiel bei der Konstruktion eines Werkzeugs zur Schlüsselverwaltung Verwendung finden kann.

```
public class CertificateInfo extends DistributionInfo
{
   public CertificateInfo (dvKeyBit keyBit, dvThingDescription certDesc,
      dvUserIdentificator ownerId);
   public dvKeyBit keyBit ();
   public DistributionInfo[] distributedKeys();
   public DistributionInfo[] distributedLocks();
   protected void registerKey(JwamKey key, dvUserIdentificator ownerId);
   protected void registerLock(JwamLock lock, dvUserIdentificator ownerId);
}
```

Code 13: Ausschnitt aus der Schnittstelle von CertificateInfo

An einem `CertificateInfo`-Exemplar lassen sich zudem die `DistributionInfos` der Schlüssel und Schlösser erfragen, die für dieses Zertifikat bereits erzeugt wurden.

5.4. Verschließbare Dinge

In Ergebnis 29 in Abschnitt 4.4.2 wurde festgehalten, daß abschließbare Gegenstände im softwaretechnischen Modell eine gemeinsame Oberklasse bekommen sollen. Diese Oberklasse stellt `LockableThingImpl` dar. `LockableThingImpl` kapselt ein Schloß und stellt die für die Zugriffskontrolle benötigten Operationen zur Verfügung.

Zur Unterstützung der Konstruktion von Behältern und Werkzeugen dienen im Rahmenwerk die Klassen `ToolImpl` bzw. `ContainerImpl`. Diese Klassen implementieren die Schnittstellen `Tool` bzw. `Collection`, `ThingAddable` und `Container`. Damit auch auf Ebene dieser Schnittstellen mit verschließbaren Behältern und Werkzeugen umgegangen werden kann, wird die Schnittstellen-Hierarchie entsprechend angepaßt. Hierzu wird die Schnittstelle `Lockable` eingeführt, welche eine Sub-Schnittstelle von `Thing` ist. So wird zum Ausdruck gebracht, daß in WAM-Systemen nur Gegenstände verschlossen werden sollen. `Tool` bzw. `Collection`, `ThingAddable` und `Container` erweitern dann die `Lockable`-Schnittstelle. Abbildung 42 stellt diese Konstruktion im Zusammenhang dar. Der Übersichtlichkeit halber wurde in dieser Abbildung auf die Darstellung des Package-Rahmens verzichtet. Es sei daher angemerkt, daß `Lockable` und `LockableThing` wie üblich im Package de.jwam.handling.accesscontrol zu finden sind.

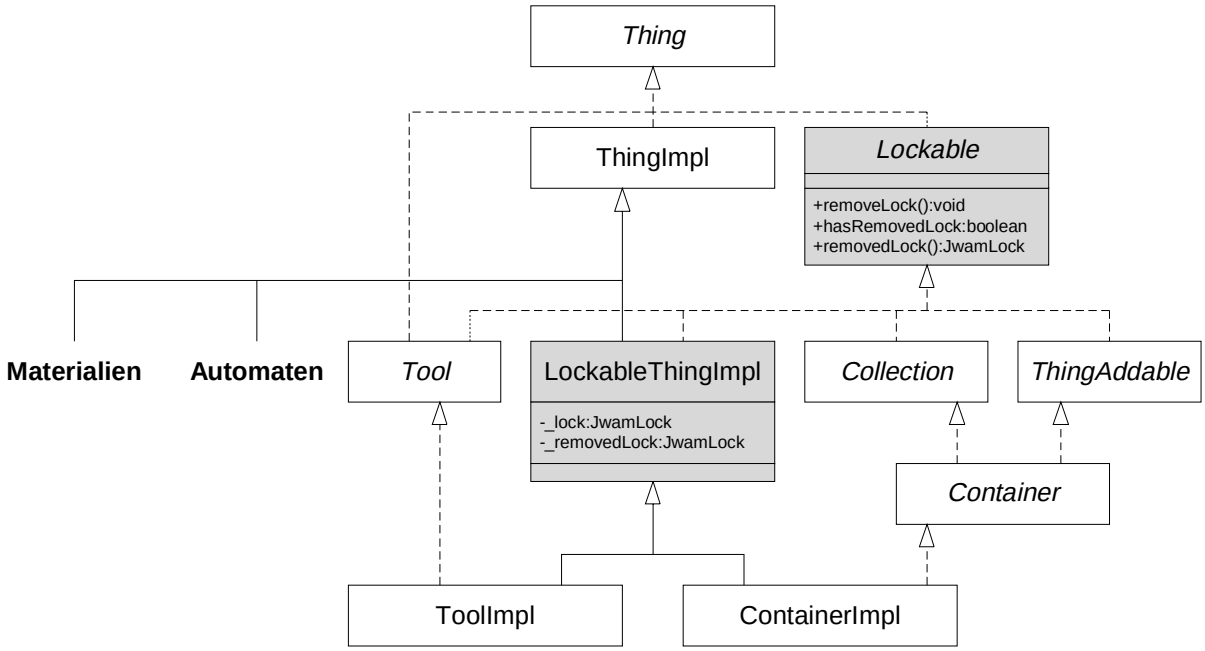


Abbildung 42: Integration von Lockable und LockableThingImpl in das JWAM-Rahmenwerk.

Die Schnittstelle der Implementationsklasse `LockableThingImpl` (siehe Code 14) ermöglicht es zu überprüfen, ob ein Schloß vorhanden ist (`hasLock()`) und ob der Gegenstand abgeschlossen ist (`isLocked()`). `isLocked()` kann auch aufgerufen werden, wenn der Behälter kein Schloß hat. In diesem Fall gibt `isLocked()` als Ergebnis `false` zurück.

```
public class LockableThingImpl extends ThingImpl implements Lockable
{
   public boolean hasLock();
   // @require !hasLock()
   // @ensure hasLock() &&!hasRemovedLock()
   public void setLock (JwamLock newLock);
   // @require hasLock() && !isLocked()
   // @ensure !hasLock && hasRemovedLock()
   public void removeLock ();
   public boolean hasRemovedLock ();
   // @require hasRemovedLock()
   public JwamLock removedLock ();
   public boolean isLocked ();
}
```

Code 14: Ausschnitt aus der Schnittstelle von LockableThingImpl

Ein abgeschlossenes Schloß wird aus Sicherheitsgründen nicht herausgegeben. Ist das Schloß unverschlossen, kann man es mittels `removeLock()` von dem verschließbaren Gegenstand entfernen und durch Aufruf von `removedLock()` erhalten. Diese Handhabung ist notwendig, da per Konvention im Rahmenwerk verändernde Operationen keinen Rückgabewert haben, und das entfernte Schloß ansonsten nicht mehr erreichbar ist.

Ein neues Schloß läßt sich mit `setLock(...)` anbringen. Dabei sollte man beachten, daß ein Schloß nicht vorhanden sein muß – ein abschließbarer Gegenstand kann eben auch über kein Schloß verfügen. Damit ist es weiterhin möglich, nicht verschlossene Werkzeuge und Behälter zu konstruieren.

5.5. Verschließbare Werkzeuge

Mit `LockableThingImpl` steht eine Klasse zur Verfügung, mit der Gegenstände – also auch Werkzeuge – prinzipiell abgeschlossen werden können. In Abschnitt 4.4 wurde beschrieben, wie sich verschlossene Werkzeuge grundsätzlich verhalten sollen. Zu klären ist aber, wie dieses Verhalten auf technischer Ebene umgesetzt werden kann.

5.5.1. Werkzeugkonstruktion in JWAM

Die Werkzeugkonstruktion im JWAM-Rahmenwerk wird hier zusammenfassend dargestellt und ausführlich in [Felski & Özkan 00] beschrieben.

FK, IAK und GUI werden im Rahmenwerk durch ein Werkzeug-Objekt umhüllt. Hierzu dient die Schnittstelle `Tool` bzw. ihre Implementationsklasse `ToolImpl`. Diese bieten unter anderem die Methoden `equip(...)` zum Ausstatten des Werkzeugs mit FK, IAK und GUI sowie `setMaterial(...)` zum Setzen des zu bearbeitenden Materials an. Ein `Tool`-Exemplar enthält keine fachliche Funktionalität, sondern verbindet die verschiedenen Bestandteile eines Werkzeugs zu einem Ganzen. `ToolImpl` war bisher eine Unterklasse von `ThingImpl`.

Ergänzend hinzu kommt das Konzept des Werkzeugkomponentenmanagers ([Felski & Özkan 00], Seite 31). Er ist nach dem Singletonmuster (s.o.) realisiert. Seine Aufgabe ist die Verwaltung aller existierenden Werkzeugkomponenten. Beim Starten einer JWAM-Applikation werden alle Werkzeuge beim Werkzeugkomponentenmanager registriert.

Soll ein Werkzeug gestartet werden, erzeugt der Werkzeugkomponentenmanager durch Verwendung der Reflection-Methode `newInstance()` ein neues Exemplar des Werkzeuges. Vereinfacht dargestellt verläuft das Starten eines Werkzeugs wie in Abbildung 43 gezeigt. Die Kanten sind mit den Operationsnamen versehen, die Zustandsübergänge auslösen. Auf die Darstellung der Operationsparameter wurde aus Gründen der Übersichtlichkeit verzichtet.

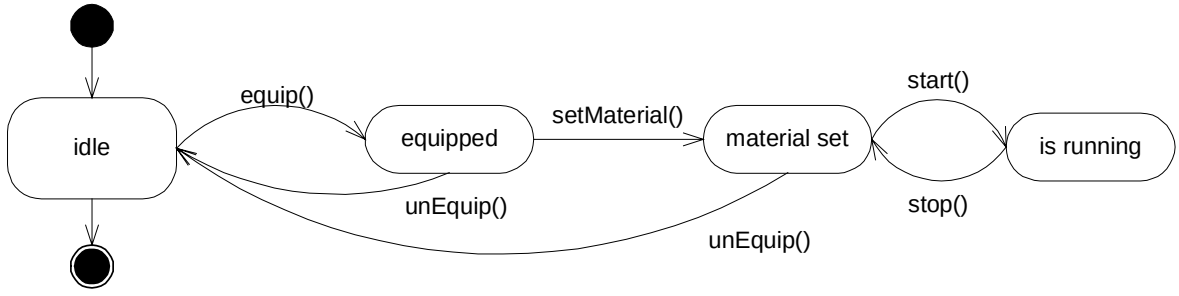


Abbildung 43: Zustandsdiagramm zum Lebenszyklus eines Werkzeugs

Nach seiner Erzeugung befindet sich ein `Tool`-Exemplar zunächst im Zustand *idle*[26]. Anschließend wird es mit FK, IAK und GUI ausgestattet, womit der Zustand *equipped* erreicht ist. Danach kann es mit einem Material versehen werden und ist dann startbereit (*material set*). Aus diesem und dem vorherigen Zustand kann aber durch Aufruf von `unEquip()` auch

[26] Um den Zusammenhang mit der Implementation in der Klasse `ToolImpl` deutlicher zu machen, werden in der Abbildung englische Bezeichnungen verwendet.

wieder der Zustand *idle* erreicht werden. Ein gestartetes Werkzeug (*is running*) kann mit `stop()`angehalten werden. Besteht keine Referenz mehr auf das Werkzeug, wird es vom Java-Garbage-Collector aus dem Speicher entfernt und damit der Endzustand erreicht.

5.5.2. Anbringen von Schlössern an Werkzeuge

Wie bereits in Abschnitt 5.4 beschrieben wurde, sollen Werkzeuge verschließbare Dinge sein. Offen bleibt bisher die Frage, wie und vor allem wann man Schlösser an das Werkzeug anbringt. Hierbei muß berücksichtigt werden, daß `Tool`-Exemplare bei Bedarf dynamisch erzeugt werden. Man kann sie – im Gegensatz zu Behältern – nicht bei Systemstart abschließen, da die entsprechenden Exemplare noch gar nicht erzeugt wurden. Ein Werkzeug ist für den Benutzer nach Aufruf der Operation `start()` verfügbar. Demzufolge muß das Werkzeug-Exemplar sein Schloß erhalten, bevor `start()` aufgerufen werden kann.

Wie bereits bekannt ist, erzeugt der Werkzeugkomponentenmanager auf Anforderung neue `Tool`-Objekte durch den Reflection-Mechanismus. Soll ein Schutzmechanismus für Werkzeuge wirksam sein, müssen die Tool-Objekte ein verschlossenes Schloß bereits erhalten haben, wenn sie vom Werkzeugkomponentenmanager zurückgegeben werden. Dies gilt für alle neu erzeugten Exemplare eines Werkzeugtyps.

Als Konsequenz wird der Werkzeugkomponentenmanager um die Methoden `lockToolType(Class toolType, JwamLock lock)` und `unLockToolType(Class toolType)` erweitert. Damit ist es möglich, alle neu erzeugten Werkzeuge eines Typs mit einem Schloß zu versehen. Dies sollte nach Möglichkeit in der Startphase eines Programmes geschehen. Wird ein Werkzeug-Typ erst zu einem späteren Zeitpunkt abgeschlossen, können in der Zwischenzeit Werkzeug-Exemplare ohne Schloß gestartet worden sein. Diese lassen sich auf die hier geschilderte Weise im nachhinein nicht mehr abschließen.

> **Ergebnis 32**
> Der Werkzeugkomponentenmanager wird dahingehend erweitert, daß Werkzeug-Typen abgeschlossen werden können. Von einem abgeschlossenen Werkzeug-Typ können nur abgeschlossene Werkzeug-Exemplare erzeugt werden.

Werkzeuge werden grundsätzlich so konstruiert, daß sie alleine lauffähig sind. Subwerkzeuge wissen nicht, ob sie von einem anderen Werkzeug umgeben sind. Aus Sicht der Zugriffskontrolle spielt es also keine Rolle, ob ein Werkzeug als einfaches, Kombi- oder Sub- Werkzeug verwendet wird. Für den Umgang mit Werkzeugen ist dies aber sehr wohl relevant, wie im folgenden verdeutlicht wird.

5.5.3. Konsequenzen für den Start von einfachen oder Kontext-Werkzeugen

Zur Klärung der Fragestellung, wie sich abgeschlossene Werkzeuge verhalten, wird nochmals der Startvorgang mit `Tool.start()` betrachtet (vgl. Abbildung 43). Offensichtlich kann `start()` nicht aus jedem Zustand heraus aufgerufen werden. Dies wurde im Rahmenwerk bisher durch die Vorbedingungen `isEquipped()`, `!isRunning()` sowie `(canStartWithoutMaterial() || hasMaterial())` ausgedrückt. Diese werden in einer Vorbedingung `canStart()` zusammengefaßt und um `!isLocked()` ergänzt.

> **Ergebnis 33**
> Ein abgeschlossenes Werkzeug kann ausgestattet und mit einem Material versehen, nicht aber gestartet werden. Dies wird durch die neue Vorbedingung `ToolImpl.canStart()` für die Methode `ToolImpl.start()` sichergestellt.

Einfache und Kontextwerkzeuge werden durch die Umgebung mit der Operation `Environment.startToolWithMaterial(Tool tool, Thing material, ...)` gestartet (vgl.

Abbildung 44). Dabei ist `tool` das zu startende Werkzeugexemplar und `material` das Material des Werkzeugs. Zunächst prüft die Umgebung mittels `Tool.canStart()`, ob das Werkzeug gestartet werden kann. Bei Aufruf dieser Operation wird die Implementationsmethode `doCanStart()` ausgeführt, in der die eigentliche Überprüfung stattfindet. Die oben genannten Bedingungen (inkl. `!isLocked()`) werden daraufhin überprüft und das Ergebnis an die Umgebung zurückgegeben. Anschließend startet sie das Werkzeug, wobei dieses nochmals `canStart()` als Vorbedingung prüft.

> **Ergebnis 34**
> Wird der Versuch unternommen, ein abgeschlossenes Werkzeug aus der Umgebung heraus zu starten, verhindert die Umgebung den Start des Werkzeuges.

In der Standardimplementation wird das Starten eines abgeschlossenen Werkzeugs demzufolge verhindert. Anwendungsentwickler haben aber die Möglichkeit, die Operationen `doCanStart()` und `doStart()` zu überschreiben, um ein anderes Verhalten zu implementieren. Anstatt gar nicht zu starten, könnte das Werkzeug dann z.B. mit einer ganz oder teilweise deaktivierten Oberfläche starten.

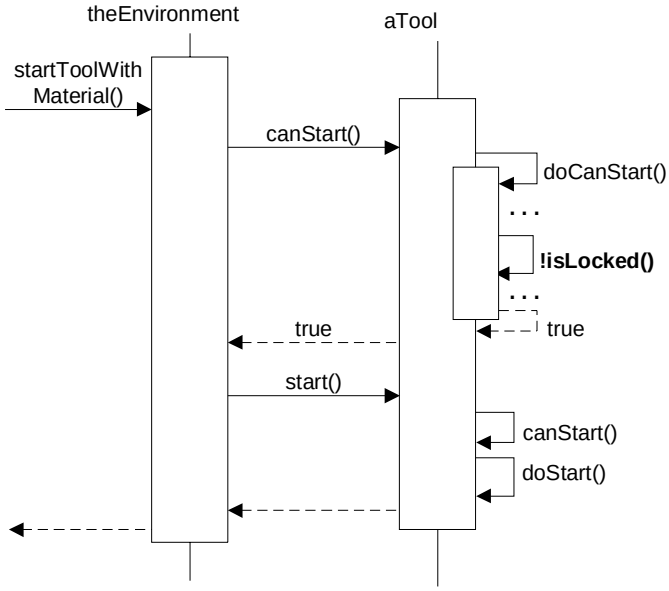


Abbildung 44: Starten eines Werkzeugs durch die Umgebung

5.5.4. Konsequenzen für den Start von Subwerkzeugen

Letztlich wird beim Start eines Subwerkzeugs genau wie bei einfachen und zusammengesetzten Werkzeugen die `start()`-Methode aufgerufen. Allerdings wird die Erzeugung eines Subwerkzeugs nicht durch die Umgebung initiiert und muß demzufolge gesondert betrachtet werden. Wenn ein Kontext-Werkzeug ein Subwerkzeug erzeugt (siehe Abbildung 45), sendet die Kontext-FK einen entsprechenden `request` zur Erzeugung einer Sub-FK. Dieser `request` wird die Zuständigkeitskette (siehe [Gamma et al. 96], Seite 367) entlang gereicht und schließlich durch das umhüllende Werkzeug-Exemplar in der Methode `ToolImpl.handleCreateFunctionality(...)` behandelt. Die benötigte Sub-FK kann nur im Zusammenhang mit einem Subwerkzeug generiert werden. Dies geschieht durch den Werkzeugkomponentenmanager mit der Methode `newToolForFunctionality(...)`.

Allerdings muß das Kontextwerkzeug berücksichtigen, daß das Erzeugen des Subwerkzeugs fehlschlagen kann und dementsprechend die Vorbedingung `canCreateToolForFunctionality(...)` prüfen. Hierbei wird das Subwerkzeug mittels `toolForFunctionality(...)`generiert und anschließend mit `canHandleMaterial()` daraufhin überprüft, ob es

das gegebene Material bearbeiten kann. Dies ist nicht der Fall, wenn das Subwerkzeug abgeschlossen ist.

Wenn Anwendungsentwickler ein abgeschlossenes Subwerkzeug starten lassen wollen, obwohl es abgeschlossen ist, müssen sie daher neben `doCanStart()` auch `doCantHandleMaterial()`überschreiben. Das Subwerkzeug könnte dann z.B. als deaktivierter Bereich im Kontextwerkzeug erscheinen.

> **Ergebnis 35**
> Soll ein abgeschlossenes Werkzeug als Subwerkzeug gestartet werden, verhindert der Werkzeugkomponentenmanager, daß das Subwerkzeug erzeugt wird. Das umgebende Kontextwerkzeug muß auf diese Möglichkeit vorbereitet sein.

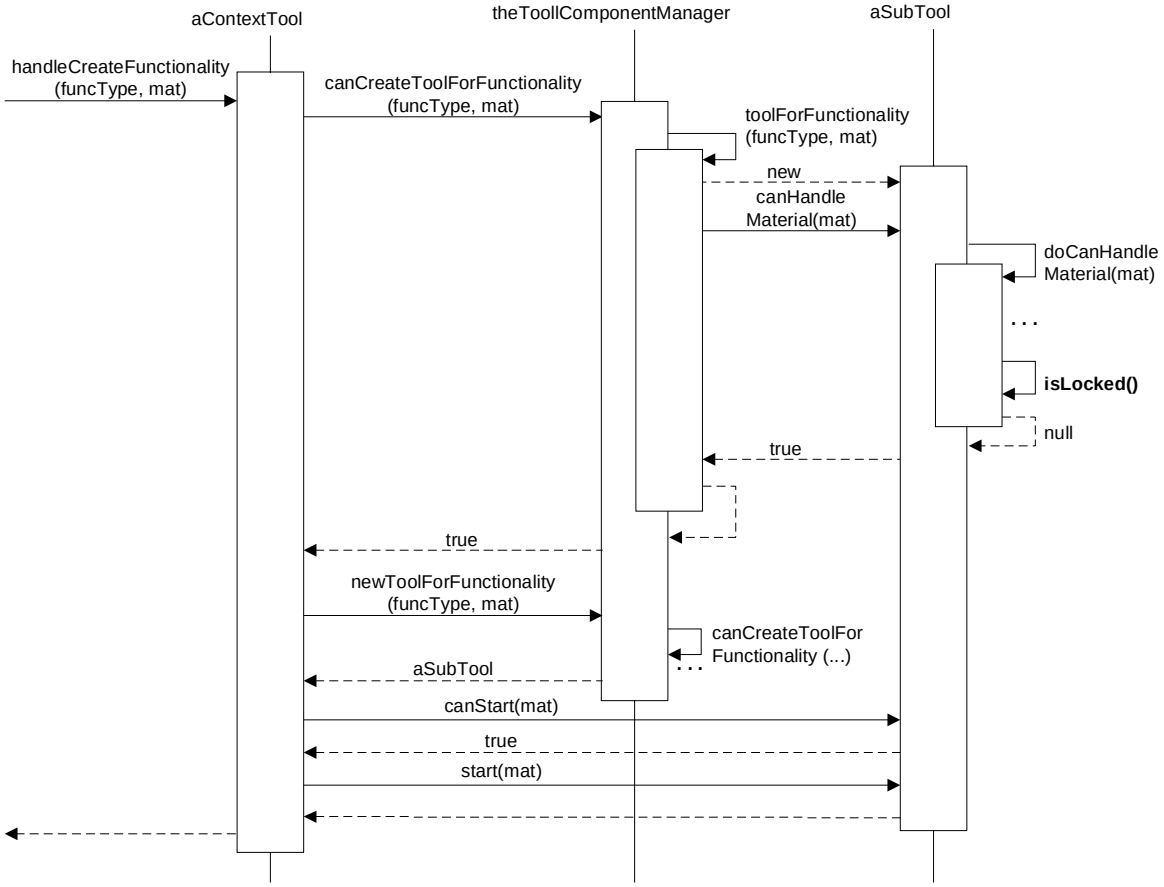


Abbildung 45: Starten eines Subwerkzeugs

5.6. Verschließbare Behälter

In Abschnitt 4.4 wurde festgehalten, daß Behälter wie Werkzeuge verschließbare Dinge sind. Damit wird `ContainerImpl` eine Unterklasse von `LockableThingImpl`. An einem verschlossenen Behälter sollen im Standardfall keine sondierenden oder verändernden Operationen ausführbar sein. Daher ist `!isLocked()` Vorbedingung nahezu jeder Methode von ContainerImpl. Lediglich `isRemovable(...)` und `isAddable(...)` können im verschlossenen Zustand noch aufgerufen werden und liefern dann als Ergebnis `false`. Damit verhalten sich alle verschlossenen Behälter wie Tresore: man kann keine Informationen über ihren Inhalt erlangen und diesen auch nicht manipulieren.

Ergebnis 36
Ein verschlossener Behälter kann nicht sondiert oder verändert werden. Dies stellt die Vorbedingung `!isLocked()` bei den entsprechenden Operationen sicher.

Durch Unterklassenbildung können aber auch einfach Behälter konstruiert werden, wo z.B. wie bei einem Briefkasten das Hinzufügen von Gegenständen weiterhin möglich ist. Die entsprechenden Operationen können hierzu einfach überschrieben und die Vorbedingungen schwächer formuliert werden. Dies ist bei Vorbedingungen durchaus legitim (siehe [Meyer 97], Seite 573).

Dies führt dazu, daß alle Werkzeuge, die auf Behältern arbeiten, entsprechend angepaßt werden müssen. Dabei sollte man auch Werkzeuge berücksichtigen, die auf bisher unverschlossenen Behältern arbeiten, denn ein Behälter kann nun jederzeit mit einem Schloß versehen und abgeschlossen werden.

Die Anpassung erfordert ein Überschreiben der bereits vorgestellten Operation `doCanHandleMaterial (Thing material)`. Diese wird nun um die Abfrage erweitert, ob der zu bearbeitende Behälter abgeschlossen ist. Wenn dies der Fall ist, läßt sich das Werkzeug nicht starten, und die Umgebung gibt eine Fehlermeldung heraus. Eine Möglichkeit, `doCanHandleMaterial (Thing material)` zu implementieren, wird in Code 15 dargestellt.

```
protected String doCanHandleMaterial (Thing material)
{
  boolean result = super.doCanHandleMaterial(material);

  if (result && material instanceof Collection)
  {
    Collection coll = (Collection) material;
    result = coll.isLocked())
  }
  return result;
}
```

Code 15: Anpassung von doCanHandleMaterial (...) auf abgeschlossene Behälter durch Überschreiben

5.7. Zusammenfassung

Die Schlüssel-Schloß-Metapher ist im Rahmen dieser Arbeit für das JWAM-Rahmenwerk konstruktiv umgesetzt worden. Der Quellcode der Konstruktion ist (bis auf die genannten Abweichungen) Bestandteil des JWAM-Rahmenwerks und befindet sich im Package de.jwam.handling.accesscontrol.

Grundlage des JWAM-Zugriffskontrollmodells sind die Klassen `User`, `JwamKey`, `JwamLock` und `JwamCertificate`. Ein `User`-Exemplar stellt das fachliche Benutzermodell dar, welches mit Schlüsseln (`JwamKey`) und Zertifikaten (`JwamCertificate`) als Privatbesitz ausgestattet werden kann. Als vereinfachte Benutzerorganisation dient `UserOrganisation`.

Die Fabrik zur Erzeugung von `JwamKey`, `JwamLock` und `JwamCertificate` wird durch `KeyLockFactory` modelliert. Bei der Beschreibung der Fabrik wurde auch darauf eingegangen, wie Informationen über die von ihr erzeugten Gegenstände gespeichert werden.

Behälter und Werkzeuge sollen entsprechend dem Entwurf als verschließbare Gegenstände modelliert werden. Als gemeinsame Oberklasse für Werkzeuge und Behälter wird die Klasse `LockableThingImpl` eingeführt, welche von `ThingImpl` erbt und ein Schloß beinhaltet. Exemplare der von `LockableThingImpl` abgeleiteten Klassen lassen sich mit einem Schloß versehen und abschließen. Das Verhalten der Klienten verschließbarer Gegenstände muß entsprechend angepaßt werden. Hierzu wurde betrachtet, wie das Starten verschlossener Werkzeuge und das Arbeiten auf verschlossenen Behältern unterbunden werden kann.

Entsprechende Eingriffe in die Konstruktion des Rahmenwerkes erlauben es, die Schlüssel-Schloß-Metapher zur Zugriffskontrolle im JWAM-Rahmenwerk zu verwenden. Da die Inte-

gration auf Ebene der Oberklassen von Werkzeugen und Behältern erfolgte, haben Anwendungsentwickler auf einfache Weise die Möglichkeit, verschließbare Behälter und Werkzeuge zu konstruieren. Lediglich selbst definierte fachliche Umgangsformen müssen mittels der Vorbedingung `!isLocked()` versehen werden, wenn der Gegenstand im abgeschlossenen Zustand nicht verwendet werden soll. Weitere Eingriffe in die Konstruktion sind nur nötig, wenn das Verhalten der Behälter und Werkzeuge vom Standardverhalten abweichen soll.

Kapitel 6 - Zusammenfassung und Ausblick

Ausgangspunkt dieser Arbeit war die Tatsache, daß es im WAM-Ansatz bisher kein fachliches Benutzermodell und damit auch keine benutzerbezogenen Rechte gab. Aus der Motivation heraus, den Zugriff von Benutzern auf anwendungsfachliche Gegenstände durch die Vergabe von Benutzerrechten zu regeln, sollte ein Zugriffskontrollmodell für den WAM-Ansatz entworfen werden.

Hierzu wurden der Begriff Zugriffskontrolle vor dem Hintergrund verschiedener Sicherheitsanforderungen definiert und die speziellen Anforderungen für Zugriffskontrollen in WAM-Systemen herausgearbeitet. Der WAM-Ansatz betont die Beteiligung von Anwendern am Entwicklungsprozeß und bestimmt durch seine Leitbilder und Entwurfsmetaphern eine bestimmte Sichtweise auf das Anwendungssystem. Ein Zugriffskontrollmodell sollte daher so gewählt werden, daß es zu den grundlegenden Konzepten des WAM- Ansatzes paßt. Als Anforderungen wurden formuliert:

- Verständlichkeit für Anwender
- Betonung der unterstützenden Sichtweise
- Hinreichende Flexibilität des Zugriffskontrollmodells im Hinblick auf verschiedene Anwendungsbereiche
- Absicherung anwendungsfachlicher Gegenstände im softwaretechnischen Modell

Bei der Betrachtung konventioneller Zugriffskontrollmodelle ergab sich, daß diese die Anforderungen nicht in befriedigender Weise erfüllen konnten. Als neuer Ansatz wurde daher untersucht, wie sich Schlüssel und Schlösser für Zugriffskontrollen in WAM-Systemen einsetzen lassen.

Im Rahmen dieser Arbeit wurde die technische Sicherheit von Anwendungssystemen nicht betrachtet. Insofern bleibt offen, wie technische Zugriffskontrollen für Zugriffe von einem WAM-System aus auf eine Ressource außerhalb des Systems erfolgen kann. Weiterhin wurde darauf hingewiesen, daß das vorgestellte Konzept auch keinen Schutz vor Angriffen von außerhalb des Systems (z.B. Manipulation einer Datenbank) bietet. Zur Abgrenzung gegenüber diesem Bereich der IT-Sicherheit wurde der Begriff *fachliche Zugriffskontrollen* eingeführt. Fachliche Zugriffskontrollen schützen anwendungsfachliche Gegenstände innerhalb eines Anwendungssystems vor nicht autorisierten Zugriffen.

Für fachliche Zugriffskontrollen in WAM-Systemen wurde die Schlüssel-Schloß-Metapher definiert und betrachtet, wie sich Schlösser in Zusammenhang mit den WAM-Entwurfsmetaphern verwenden lassen. Dabei sollen Schlüssel die fachlichen Rechte der Anwender modellieren und Schlösser dienen als Zugriffskontrollen bei den abgesicherten Objekten. Behälter, Werkzeuge, Automaten, fachliche Services und der Arbeitsplatz eines Benutzers (als Bestandteil der Umgebung) sollen sich mit Schlössern absichern lassen. Materialien hingegen können nicht mit Schlössern versehen werden, sondern werden bei Bedarf in abschließbaren Behältern aufbewahrt.

Schlüssel und Schlösser werden in der Metapher prinzipiell als Sicherheitsschlüssel bzw. –schlösser betrachtet. Schlüssel, Schlösser und Zertifikate werden durch einen fachlichen Service hergestellt. Ein Anwender kann diesen nutzen und einen Schlüssel oder ein Schloß herstellen, wenn er Eigentümer eines entsprechenden Zertifikates ist.

Anwender können Schlüssel untereinander austauschen. Auf diese Weise enthält die Schlüssel-Schloß-Metapher Ansätze für eine Vertreterregelung, da ein Anwender einen Schlüssel bei Bedarf einem anderen Anwender zukommen lassen kann.

Aus der Betrachtung verschiedener Leitbilder sowie der Berücksichtigung kooperativer Arbeitssituationen ergaben sich neue Einflüsse auf die Schlüssel-Schloß-Metapher. Es wurde

darauf hingewiesen, daß sie durch ihre einfache Handhabbarkeit prinzipiell an jedem Arbeitsplatz einsetzbar ist, jedoch in der Modellierung unterschiedlich deutlich in Erscheinung tritt.

Beim Entwurf eines softwaretechnischen Modells für Schlüssel, Schlösser und Zertifikate wurde das Konzept des Schlüsselbartes betrachtet und begründet, warum dieser als Fachwert modelliert werden kann. Ein Schlüsselbart ist einem Zertifikat zugeordnet und alle erzeugten Schlüssel und Schlösser erhalten diesen Schlüsselbart als Bestandteil. Der Schlüsselbart ist in der Lage, hierarchische Sub-Schlüssel/Super-Schlüssel-Strukturen abzubilden. Auf dieser Grundlage kann ein Schloß dann ermitteln, ob ein Schlüssel zu ihm paßt. Dies ist der Fall, wenn die Schlüsselbärte gleich sind oder der Schlüsselbart des Schlüssels dem des Schlosses hierarchisch übergeordnet ist.

Anschließend wurde untersucht, wie sich Gegenstände auf Entwurfsebene mit Schlössern absichern lassen. Es zeigte sich, daß es sinnvoll ist, eine gemeinsame Oberklasse für verschließbare Gegenstände einzuführen, die ein Schloß beinhaltet. Abschließbare Gegenstände stellen durch Vorbedingungen in den entsprechenden Methoden sicher, daß sie im abgeschlossenen Zustand nicht benutzt werden können. Die Anwendung dieses Konzeptes auf Behälter, Werkzeuge und Subwerkzeuge wurde beschrieben. Offen bleibt hingegen, wie die Unterstützung zum Schutz von Automaten und fachlichen Services auf Entwurfsebene aussehen kann.

Der Vergleich dieses Entwurfes mit dem Java-Zugriffskontrollmodell ergab, daß sich die Schlüssel-Schloß-Metapher als benutzerbezogenes Berechtigungskonzept nur unter erheblichem technischen Aufwand auf das Java-Zugriffskontrollmodell abbilden läßt. Unbeschadet dessen kann das Java-Sicherheitsmodell verwendet werden, um die Anwendungssicherheit auf technischer Ebene zu erhöhen.

Auf Basis des Entwurfes erfolgte im Rahmen dieser Arbeit die Implementation der Schlüssel-Schloß-Metapher und ihre Integration in das JWAM-Rahmenwerk. Die Fabrik zur Herstellung von Schlüsseln, Schlössern und Zertifikaten wurde dabei in einer vereinfachten Variante realisiert. Offen bleibt, wie sich diese Fabrik als echter fachlicher Service realisieren läßt.

Bei der Beschreibung der Konstruktion wurde herausgearbeitet, wie Zugriffskontrollen mit Schlüsseln und Schlössern in das Rahmenwerk eingefügt wurden. Als Oberklasse für verschließbare Gegenstände wurde die Klasse `LockableThingImpl` eingeführt. Die Behälter- und Werkzeug- Konstruktion wurde entsprechend umgestaltet, so daß Werkzeuge und Behälter im Rahmenwerk jetzt generell verschließbar sind.

Das Anbringen von Schlössern an Behälter erfolgt beim Systemstart. Hierfür haben Anwendungsentwickler Sorge zu tragen. Aufgrund der Besonderheiten bei der Werkzeugkonstruktion können Werkzeuge jedoch nicht zum Systemstart mit Schlössern versehen werden. Werkzeuge werden durch den Werkzeugkomponentenmanager erzeugt. Dieser wurde daher so angepaßt, daß nun Werkzeugtypen abgeschlossen werden können. Ist ein Werkzeugtyp abgeschlossen, erzeugt der Werkzeugkomponentenmanager lediglich abgeschlossene Werkzeugexemplare.

Im Hinblick auf den Umgang mit verschlossenen Gegenständen ist eine Anpassung der Komponenten notwendig, die mit den verschlossenen Gegenständen umgehen. Betrachtet wurden hierbei wiederum keine Automaten und fachlichen Services. Handelt es sich um einen verschlossenen Behälter, kann ein Werkzeug nicht auf diesem arbeiten. Bei verschlossenen Werkzeugen ist zu unterscheiden, ob es sich um ein Subwerkzeug oder ein Kontextwerkzeug handelt. Der Start von abgeschlossenen Kontextwerkzeugen wird durch die Umgebung verhindert. Abgeschlossene Subwerkzeuge werden durch den Werkzeugkomponentenmanager gar nicht erst zurückgegeben, so daß sie durch das Kontextwerkzeug nicht dargestellt werden können.

Bei diesen Vorgehensweisen handelt es sich um den Standard-Umgang mit verschlossenen Gegenständen, wie er im Rahmen dieser Arbeit für das Rahmenwerk vorgesehen und integriert wurde. Anwendungsentwickler haben auf einfache Weise die Möglichkeit, konkrete verschließbare Gegenstände zu konstruieren. Es wurde ebenfalls darauf hingewiesen, wie sich das beschriebene Standardverhalten ändern läßt, so daß z.B. aus einem verschlossenen Behälter nichts entnommen, wohl aber etwas hineingelegt werden kann.

Das fachliche Benutzermodell wird im Rahmen dieser Arbeit hauptsächlich benötigt, um Anwender mit Zugriffsrechten versehen zu können. Das fachliche Modell eines Benutzers umfaßt demnach zumindest eine Repräsentation des Benutzers (z.B. in Form eines Namens) sowie je einen fachlichen Behälter für Schlüssel und Zertifikate.

Technisch motiviert kommen eine Benutzerkennung und ein Paßwort sowie ein Modell eines Standard-Benutzers hinzu. Verschiedene andere Konzepte innerhalb des WAM-Ansatzes benötigen einen fachlichen Verweis auf den Benutzer. Dieser wird in der Konstruktion durch den Fachwert `dvUserIdentificator` modelliert, wobei dieser Fachwert zur Repräsentation die Benutzerkennung und zur technischen Identifikation einen Fachwert `dvIdentificator` enthält.

Zur Verwaltung von Benutzern wird ein Werkzeug zur Benutzerverwaltung sowie ein materialverwaltender, fachlicher Service benötigt, der die Benutzermodelle organisiert (die *Benutzerorganisation*). Eine vereinfachte Implementation dieses Service erfolgte im Rahmen der Arbeit. Offen bleibt, wie die Benutzerorganisation als echter fachlicher Service mit Anschluß an ein Persistenzmedium (z.B. einen Directory Server) entworfen und konstruiert werden kann.

Abschließend läßt sich somit sagen, daß es gelungen ist, mit der Schlüssel-Schloß-Metapher ein Modell zu beschreiben und konstruktiv umzusetzen, welches für Zugriffskontrollen in WAM-Anwendungssystemen geeignet ist.

Literaturverzeichnis

[Adams, Sasse 99]
Anne Adams and Martina Sasse: „Users are not the enemy“, Communications of the ACM, December 1999, Vol. 42, No. 12

[Andreae 93]
T. Andreae: „Mathematik für Studierende der Informatik“ Scriptum zur gleichnamigen Vorlesung, Universität Hamburg, Fachbereich Mathematik, Hamburg, 1993

[Barkley 97]
J.F. Barkley: „Comparing Simple Role Based Access Control Models and Access Control Lists“ in: Proceedings of the 2. ACM Workshop on Role-Based Access Control, ACM Press, New York, November 1997
siehe auch http://csrc.nist.gov/rbac (19.11.00)

[Bäumer 98]
Dirk Bäumer: „Softwarearchitekturen für die rahmenwerkbasierte Konstruktion großer Anwendungssysteme“, Dissertationsschrift am Fachbereich Informatik, Universität Hamburg, 1998

[BDSG]
Bundesdatenschutzgesetz vom 20. Dezember 1990 (BGBl. I S.2954)
http://www.datenschutz-berlin.de/recht/de/bdsg/bdsg1.htm (19.11.00)

[Beis, Vorwerk 97]
C. Beis, S.H. Vorwerk: „Chipkartengestütztes, rollenbasiertes Zugangssystem im Hotelgewerbe mit Anbindung an weitere Anwendungen“, Studienarbeit am Fachbereich Informatik, Universität Hamburg, 1997

[BGB]
Bürgerliches Gesetzbuch vom 18. August 1986 1990 (BGBl. I S.195)

[Bleek 97]
Wolf Gideon Bleek: „Techniken zur Konstruktion verteilter und technisch eingebetteter Anwendungssysteme“ , Diplomarbeit am Fachbereich Informatik, Universität Hamburg, 1997

[Bleek et al. 99a]
W.-G. Bleek , M. Lippert, S. Roock, W. Strunk , H. Züllighoven: „Frameworkbasierte Anwendungsentwicklung (Teil 3) – die Anbindung von Benutzungsoberflächen und Entwicklungsumgebungen an Frameworks“, in: OBJEKTspektrum 3/1999

[Bleek et al. 99b]
W.-G. Bleek , A. Havenstein, S. Roock, I. Wetzel, H. Züllighoven: „Frameworkbasierte Anwendungsentwicklung (Teil 4) – Fachwerte“, in: OBJEKTspektrum 5/1999

[Bohlmann 98]
Holger Bohlmann: „Realisierung einer Behälterbibliothek in Java – Die jConLib“, Studienarbeit am Fachbereich Informatik, Universität Hamburg, 1999

[Booch et al. 99]
G. Booch, I. Jacobson, J. Rumbaugh: „The Unified Modeling Language User Guide“, Addison-Wesley, Reading, Massachusetts, 1999

[BSI 97]
Gerhard Banse: „Nichttechnisches in der IT-Sicherheit“, Bundesamt für Sicherheit in der Informationstechnik, 1997
http://www.bsi.bund.de/literat/tagungsb/banse.htm (19.11.00)

[Chang, Jiang 89]
Carl K. Chang, Tasng Ming Jiang: „A binary Single-Key-Lock System for Access Control“, IEEE Transactions on Computers, Vol. 38, No. 10, October 1989

[DoD 1985]
United States of America, Department of Defence: „Trusted Computer Systems Evaluation Criteria (TCSEC)“, DoD 5200.28-STD, Dezember 1985

[dtv Lexikon 90]
dtv Lexikon in 20 Bänden, F.A. Brockhaus GmbH (Mannheim) & Deutscher Taschenbuch Verlag (München), 1990

[Encarta 99]
Microsoft® Encarta® Enzyklopädie 2000, © 1993-1999 Microsoft Corporation

[Felski & Özkan 00]
Johanna Felski, Erdal Özkan: „Rahmenwerkbasierte Werkzeugkomponenten am Beispiel des JWAM-Rahmenwerks“, Studienarbeit am Fachbereich Informatik, Universität Hamburg, 2000

[Felten, McGraw 98]
Ed Felten, Garry McGraw: „Securing Java“, Wiley and Sons, Chichester (UK), New York, Brisbane, 1998

[Ferraiolo, Cugini, Kuhn 1995]
D.F. Ferraiolo, J.A. Cugini, D.R. Kuhn: „Role Based Access Control (RBAC): Features and Motivation“, 11. Annual Computer Security Applications Conference ACSAC 1995, New Orleans, IEEE Computer Society Press, Los Alamitos, Dezember 1995, S. 241 – 248
siehe auch http://csrc.nist.gov/rbac (19.11.00)

[Ferraiolo, Gilbert, Lynch 93]
D.F. Ferraiolo, D.M. Gilbert, N. Lynch: „An examination of federal and commercial access control policy needs“, Proceedings of the 16. NIST National Computer Security Conference, NCSC 1993, Baltimore, MD, 1993, S. 107 – 116
siehe auch http://csrc.nist.gov/rbac (19.11.00)

[Ferraiolo, Kuhn 92]
D.F. Ferraiolo, D.R. Kuhn: „Role Based Access Control“, Proceedings of the 15. NIST National Computer Security Conference NCSC 1992, Baltimore, MD, October 1992, S. 554 – 563
siehe auch http://csrc.nist.gov/rbac (19.11.00)

[Flanagan 99]
David Flanagan: „Java in a Nutshell“, O’Reilly, Sebastopol, CA, 2000

[Floyd 94]
C. Floyd: „Software-Engineering – und dann?“, Informatik Spektrum, Band 17, Heft 1, Springer-Verlag, Berlin, Heidelberg, New York, Tokio, 1994, S. 29-37, 1994

[Floyd 99]
C. Floyd, G. Gryczan, J. Mack: „Einführung in die Softwaretechnik“, Scriptum zur gleichnamigen Vorlesung, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik, Hamburg, 1999
http://swt-www.informatik.uni-hamburg.de/teaching/ws99-2000/est (19.11.00)

[Freund 00]
Mirko Freund: „Komponenten zur Kooperationsunterstützung: Das Postversandsystem“, Diplomarbeit am Fachbereich Informatik, Universität Hamburg, 2000

[Gamma et al. 96]
E. Gamma, R. Helm, R. Johnson, J. Vlissides: „Entwurfsmuster - Elemente wiederverwendbarer Software“, dt. Übersetzung von D. Riehle, Addison Wesley, Bonn, 1996

[Gryczan 96]
G. Gryczan: „Prozeßmuster zur Unterstützung kooperativer Tätigkeit“, Deutscher Universitätsverlag, Wiesbaden, 1996

[Gryczan et al. 99a]
G. Gryczan, C. Lillienthal, M. Lippert, S. Roock, H. Wolf, H. Züllighoven: „Frameworkbasierte Anwendungsentwicklung (Teil 1)“, in: OBJEKTspektrum 1/1999

[Gryczan et al. 99b]
[Gryczan et al. 99b] Guido Gryczan, Andreas Havenstein, Stefan Roock, Ingrid Wetzel, Heinz Züllighoven: „Frameworkbasierte Anwendungsentwicklung (Teil 2) – die Konstruktion interaktiver Anwendungen“, in: OBJEKTspektrum 2/1999

[Gryczan et al. 99c]
G. Gryczan, A. Krabbel, I. Wetzel, H. Züllighoven: „Application-Oriented Software Development for Supporting Cooperative Work“, erschienen in: „Human-Computer Interaction. Ergonomics and User Interfaces“, Volume1, edited by Hans-Jörg Bullinger, Jürgen Ziegler. S. 1213-1217, 1999

[Gryczan et al. 00a]
[Gryczan et al. 00a] G. Gryczan, C. Lillienthal, M. Lippert, S. Roock, H. Wolf, H. Züllighoven: „Frameworkbasierte Anwendungsentwicklung (Teil 5) – Unterstützung von Kooperation mit persistenten fachlichen Behältern“, in: OBJEKTspektrum 1/2000

[Gryczan et al. 00b]
[Gryczan et al. 00b] G. Gryczan, A. Havenstein, S. Roock, I. Wetzel, H. Züllighoven, „Frameworkbasierte Anwendungsentwicklung (Teil 6): Frameworkentwicklung und -einsatz organisieren“, in: OBJEKTspektrum 2/2000

[Havenstein 99]
A. Havenstein: „Unterstützung Kooperativer Arbeit durch eine Software-Registratur”, Diplomarbeit am Fachbereich Informatik, Universität Hamburg, 2000

[Henke 00]
Stefan Henke: „Biometrische Authentisierungsverfahren“, Diplomarbeit am Fachbereich Informatik, Universität Hamburg, 2000

[Hoare 69]
C.A.R. Hoare: „An Axiomatic Base for Computer Programming“, Communications of the ACM, October 1969, Vol. 12, No. 10, pp 576-583

[Hwang, Shao, Wang 92]
J.J. Hwang, B.M. Shao und P.C. Wang: „A new Access Control Method Using Prime Factorisation“, The Computer Journal; 35(1): 16-20, 1992
siehe auch http://www3.oup.co.uk/computer_journal/hdb/Volume_35/Issue_01 (12.09.00)

[ITSEC 91]
Amt für amtliche Veröffentlichungen der europäischen Gemeinschaft: „Kriterien für die Bewertung der Sicherheit von Systemen der Informationstechnik (ITSEC)“, Version 1.2, Luxemburg, 1991

[Jacobson 92]
I. Jacobson: „Object-Oriented Software Engineering – A Use Case Driven Approach“ , Addison-Wesley, Reading, Massachusetts, 1992

[Jain et al. 00]
Anil Jain, Lin Hong, Sharata Pankanti: „Biometric Identification“, Communications of the ACM, Vol. 43, No. 2, February 2000

[JWAM 00]
„JWAM: Java Framework for the Tools and Materials Approach“, http://www.jwam.de (19.11.00)

[Kessler, Mund 93]
Volker Kessler, Sibylle Mund: „Sicherheitsmodelle“, Studie im Rahmen des Verbundprojektes REMO (Referenzmodell für sichere IT-Systeme) am Europäischen Institut für Systemsicherheit der Universität Karlsruhe (1993), Siemens, München

[Koch 00]
Jörn Koch: „JWAM-Framework-Komponenten zur Kooperationsunterstützung (COJAC): Realisierung eines Raumkonzepts“, Diplomarbeit am Fachbereich Informatik, Universität Hamburg, 2000

[Lehman 80]
M.M. Lehman: „Programs, Life Cycles, and Laws of Software Evolution“, Proc. of IEEE, Vol 68, No. 9, Sept. 1980, pp. 1060–1076.

[Lippert 99]
Martin Lippert: „Die Desktop-Metapher in Systemen nach dem Werkzeug- und Material-Ansatz“, Diplomarbeit am Fachbereich Informatik, Universität Hamburg, 1999

[MSDN 99a]
Ruediger R. Asche: „The Guts of Security“ in Microsoft Developer Network, Microsoft Corporation, Redmond, 1999
http://msdn.microsoft.com/library/techart/msdn_secguts.htm (18.11.00)

[MSDN 99b]
Microsoft Developer Network Library, Microsoft Corporation, Redmond, 1999
http://msdn.microsoft.com (18.11.00)

[Meyer 97]
B. Meyer: „Object-Oriented Software Construction”, Second Edition, Prentice-Hall, New York, London, 1997

[Müller 99]
Klaus Müller: „Konzeption und Umsetzung eines Fachwertkonzepts“, Studienarbeit am Fachbereich Informatik, Universität Hamburg, 1999

[Oppliger 97]
R. Oppliger: „IT-Sicherheit – Grundlagen und Umsetzung in der Praxis“, Vieweg Verlag, 1997

[Otto, Schuler 00]
Michael Otto, Norbert Schuler: „Fachliche Services: Geschäftslogik als Dienstleistung für verschiedene Benutzungsschnittstellen-Typen“, Diplomarbeit am Fachbereich Informatik, Universität Hamburg, 2000

[Pfleeger 97]
Charles Pfleeger: „Security in Computing“, 2. Auflage, Prentice Hall, 1997

[Rankl, Effing 1996]
W. Rankl, W. Effing: „Handbuch der Chipkarten – Aufbau, Funktionsweise, Einsatz“, 2. Auflage, Carl Hanser Verlag, München, 1996

[Roock & Wolf 98]
Stefan Roock, Henning Wolf: „Die Raummetapher zur Entwicklung kooperationsunterstützender Softwaresysteme für Organisationen“ Diplomarbeit am Fachbereich Informatik, Universität Hamburg, 1998

[Schier99]
Kathrin Schier: „Vertrauenswürdige Kommunikation im elektronischen Zahlungsverkehr – ein formales Rollen- und Aufgabenbasiertes Sicherheitsmodell für Anwendungen mit multifunktionalen Chipkarten“, Dissertationsschrift am Fachbereich Informatik, Universität Hamburg,1999

[Silberschatz 98]
P.B. Galvin, A. Silberschatz: „Operating System Concepts”, 5th Edition, Addison Wesley, Reading, Massachusetts, 1998

[Sohr 00]
Karsten Sohr: „Sandkastenspiele“, c’t – Magazin für Computertechnik, Heft 11, Seite 226-232, Verlag Heinz Heise, Hannover, 2000

[Sun 96]
Sun Microsystems, Inc.: „The Java Language Environment – A White Paper“, 1996 http://java.sun.com/docs/white/langenv (8.11.00)

[Sun 00a]
Sun Microsystems Inc.: „The Java Development Kit 1.3“, 2000 http://java.sun.com/products/jdk/1.3 (12.09.00)

[Sun 00b]
Sun Microsystems, Inc.: „Java Security Architecture“, 2000 http://java.sun.com/j2se/1.3/docs/guide/security/spec/security-specTOC.fm.html (12.09.00)

[Szyperski 97]
Clemens Szyperski: „Component Software – Beyond Object-Oriented Programming“, Addison-Wesley, Reading, Massachusetts, 1997

[Tanenbaum 95]
Andrew S. Tanenbaum: „Moderne Betriebssysteme”, 2. Auflage, Coedition der Verlage Carl Hanser (München) und Prentice-Hall International, Englewood – Cliffs, New Jersey, 1995

[Traub 95]
Horst-Peter Traub: „Objektorientierte Behälterklassen-Bibliotheken“, Diplomarbeit am Fachbereich Informatik, Universität Hamburg, 1995

[Weske, Wulf 94]
Dirk Weske, Martina Wulf: „Konzepte zur Materialversorgung verteilter Werkzeugumgebungen am Beispiel der Anbindung einer objektorientierten Datenbank“, Diplomarbeit am Fachbereich Informatik, Universität Hamburg, 1994

[Wu, Hwang 84]
M. Wu, T. Hwang: „Access control with single-key-lock“, IEEE Transactions on Software Engeneering, Vol. SE-10, No. 2, pp. 185-191, March 1984

[Wulf 95]
Martina Wulf: „Konzeption und Realisierung einer Umgebung zur Koordination rechnergestützter Tätigkeiten in kooperativen Arbeitsprozessen“, Diplomarbeit am Fachbereich Informatik, Universität Hamburg, 1995

[Züllighoven et al. 98]
H. Züllighoven et al.: „Das objektorientierte Konstruktionshandbuch nach dem Werkzeug & Material-Ansatz“, Heidelberg, dpunkt-Verlag, 1998

[Züllighoven et al. 01]
überarbeitete, englischsprachige Version von [Züllighoven et al. 98] (in Vorbereitung, voraussichtliche Veröffentlichung: 2001)

Abbildungsverzeichnis