Entwurf und Realisierung eines Webportals für den Zugriff auf Patientenleitfäden mit Hilfe von Ercatons

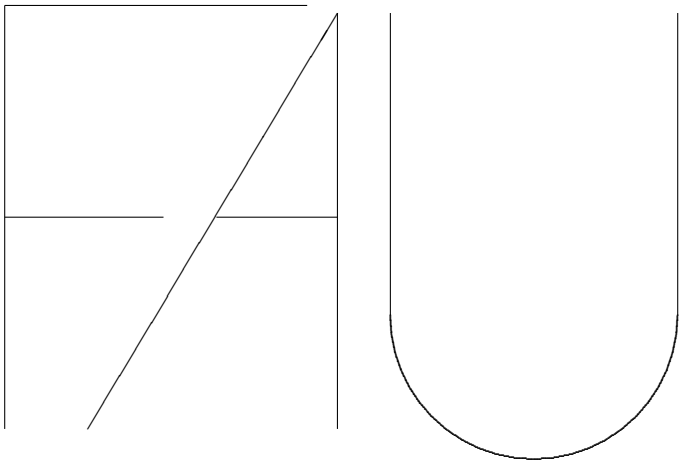
Studienarbeit

Manuela Schinn

Lehrstuhl für Informatik 6
(Datenmanagement)

Department Informatik
Technische Fakultät

Friedrich Alexander-
Universität
Erlangen-Nürnberg



Entwurf und Realisierung eines Webportals für den Zugriff auf Patientenleitfäden mit Hilfe von Ercatons

Studienarbeit im Fach Informations- und Kommunikationstechnik

vorgelegt von

Manuela Schinn

geb. 31.08.1980 in Kelheim

angefertigt am

Department Informatik
Lehrstuhl für Informatik 6 (Datenmanagement)
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer: Prof. Dr. Richard Lenz
Dipl.-Inf. Christoph P. Neumann
Dipl.-Inf. Florian Irmert

Beginn der Arbeit: 01.08.2009
Abgabe der Arbeit: 05.03.2010

Erklärung zur Selbständigkeit

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass diese Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Der Universität Erlangen-Nürnberg, vertreten durch den Lehrstuhl für Informatik 6 (Datenmanagement), wird für Zwecke der Forschung und Lehre ein einfaches, kostenloses, zeitlich und örtlich unbeschränktes Nutzungsrecht an den Arbeitsergebnissen der Studienarbeit einschließlich etwaiger Schutzrechte und Urheberrechte eingeräumt.

Erlangen, den 05.03.2010

(Manuela Schinn)

Kurzfassung

Entwurf und Realisierung eines Webportals für den Zugriff auf Patientenleitfäden mit Hilfe von Ercatons

Diese Studienarbeit befasst sich mit Ercatons, eine Variante des Prototyp-basierten Programmiersprachenansatzes. Sie wurden von der Living Pages Research GmbH entwickelt und implementieren das Konzept der organischen Programmierung (OrP), ein Ansatz der versucht die Grenzen der objektorientierten Programmierung zu überschreiten. Es werden die Grundlagen der Prototyp-basierten Programmierung erörtert und die Programmiersprachen Self und JavaScript als Beispiel hierfür erläutert. Um die Idee der OrP zu verdeutlichen, wird eine Einführung in das sogenannte *Thing* gegeben und darauf aufbauend die OrP besprochen und zur Prototyp-basierten Programmierung abgegrenzt. Ercatons dienen als technische Infrastruktur für die OrP und werden ebenfalls vorgestellt. Es werden ihre Spezifikation, die zugrunde liegende virtuelle Maschine und die verschiedenen Zugriffsarten erläutert. Mit Hilfe von Ercatons wird in dieser Arbeit ein Webportal zum Zugriff auf Patientenleitfäden entwickelt. Als Vorarbeit wird hierzu der Software-Entwurf mittels Ercatons betrachtet und sowohl ausgewählte Konzepte der Ercatons, wie Vererbung, mehrwertige Attribute und Aktionen, als auch Konzepte zur Zugriffskontrolle, Persistenz, Abfrage und Anzeige untersucht. Eine Anforderungsanalyse für das Web Portal, Entwurf und Implementierung sind Bestandteile des folgenden Kapitels. Zusätzlich werden die Vor- und Nachteile der Realisierung des Webportals mit Ercatons gegenüber der traditionellen Software-Entwicklung aufgezeigt. Ein Epilog mit Zusammenfassung, zukünftigen Aufgaben und einem Resume runden die Studienarbeit ab.

Abstract

Design and Realization of a Web Portal for Access to Patient Guidelines by Means of Ercatons

This thesis introduces Ercatons, a version of the prototype-based programming language approach. It was developed by Living Pages Research GmbH and implements the concept of Organic Programming (OrP), a programming model which tries to overcome the limitations of object-oriented programming.
The basics of prototype-based programming will be explained whereas the programming languages Self and JavaScript serve as examples. In order to clarify the idea of OrP, the so called *Thing* will be introduced. With its help the concepts of OrP will be explained and compared to prototype-based programming. Ercatons serve as technical infrastructure for OrP and will be presented as well. Their specifications, the underlying virtual machine as well as the different ways to access Ercatons are evaluated. A Web Portal for access to patient guidelines will be designed and realized by means of Ercatons. In order to prepare this, important software design concepts with Ercatons like inheritance, multivalued attributes and actions, as well as concepts for access control, persistence, query and display will be observed. A requirements analysis for the Web Portal, design and implementation are part of the following chapter. Additionally, advantage and disadvantage of developing this Web Portal with Ercatons compared to traditional software-design will be discussed. An epilogue with summary, future work and resume completes this thesis.

Vorwort und Danksagung

Mein besonderer Dank gilt Falk Langhammer von der Living Pages Research GmbH, der mich während der gesamten Studienarbeit per E-Mail mit Informationen zu den Ercatons versorgt hat. Er hat geduldig, schnell und ausführlich meine Fragen beantwortet, zu jeder Tag- und Nachtzeit.
Außerdem möchte ich Christoph Neumann und Florian Irmert für ihre Betreuung am Lehrstuhl danken. Vielen Dank auch an Diana und Ralf für das Korrekturlesen dieser Studienarbeit und an meine Eltern, die mich in der Schreibphase bei sich aufgenommen und mit Essen und Trinken versorgt haben. Großer Dank natürlich auch an meinen Freund Thilo, der mich seelisch unterstützt hat und an so manchem Tag entbehren musste.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

OrP	organische Programmierung
OOP	objektorientierte Programmierung
SW-System	Software-System
API	Application Programming Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
SOAP	Simple Object Access Protocol
URL	Uniform Resource Locator
WWW	World Wide Web
XML	Extended Markup Language
XSLT	XSL Transformation
EJBs	Enterprise JavaBeans
EJB	Enterprise JavaBean
Java EE	Java Enterprise Edition
JPEG	Joint Photographic Experts Group
GIF	Graphics Interchange Format

ESH	Ercato Shell
DOM	Document Object Model
AWMF	Arbeitsgemeinschaft der Wissenschaftlichen Medizinischen Fachgesellschaften
MDA	Model Driven Architecture

1 Einleitung

Diese Studienarbeit befasst sich mit Ercatons, einer Variante des Prototyp-basierten Programmiersprachenansatzes. Sie wurden von der Living Pages Research GmbH entwickelt, inspiriert dadurch, wie mit Gegenständen der realen Welt umgegangen wird und wie die traditionellen objektorientierten Techniken von diesem Ideal abgewichen sind. Ercatons implementieren das Konzept der organischen Programmierung (OrP), ein Ansatz der versucht, die Grenzen der objektorientierten Programmierung (OOP) zu überschreiten [ILvW05]. Es wurde bereits einige Jahre erfolgreich mit Ercatons entwickelt, wobei unter anderem ein großes Projekt für Henkel KGaA (Düsseldorf, Deutschland) entstanden ist. Das System wird zur Entwicklung und teilweise zur Produktion neuer chemischer Rezepte verwendet. Seit 2003 ist es produktiv und wird ständig weiter entwickelt. Es hat mittlerweile eine hohe Komplexität erreicht, wobei es immer noch stabil ist und eine hohe Benutzerakzeptanz hat [ILvW05].

Motivation

Die OrP baut auf dem Konzept eines Dings (Fachterm im Englischen: *Thing*) auf. Ein *Thing* in einem Software-System (SW-System) verhält sich wie ein autonomes Objekt. Software-Objekte machen dies nicht, weshalb es nötig ist, ein traditionelles SW-System im Voraus zu modellieren, anstatt es im laufenden Betrieb Stück für Stück aufzubauen und weiter zu entwickeln. Besonders in Großprojekten kann die traditionelle Vorgehensweise zu Problemen führen, da es oft unmöglich ist ein komplexes SW-System vollständig und fehlerfrei zu modellieren. In [ILvW05] wird als Beispiel für ein SW-System das WWW genannt. Es wurde nach und nach aufgebaut und Änderungen werden nur inkrementell getätigt, das heißt zum Beispiel durch Hinzufügen von Text zu einer bestimmten Webseite.

Kurzcharakteristik

Der Nachteil der Klassen-orientierten Programmierung besteht darin, dass Klassen nicht inkrementell wachsen können. Die Prototyp-basierte Programmierung versucht mit Hilfe von Prototypen dieses Limit zu überkommen. Jedoch reicht auch dieses Konzept noch nicht aus. Objekte der Prototyp-basierten Programmiersprachen, wie Self oder Java Script, sind nicht persistent und haben keine eigene Anzeige. Die Idee der OrP ist es, diese Grenzen zu überwinden. Ein Thing besitzt ein Aussehen und bestimmte Eigenschaften durch die es beschrieben wird. Es hat eine eigene Präsentationsschicht und ist von Grund auf persistent.

Ziel der Arbeit

Das Ziel dieser Studienarbeit ist das Einarbeiten in Ercatons, eine technische Infrastruktur für OrP. Außerdem ist die Untersuchung des Ansatzes der OrP und dessen Abgrenzung gegen Prototyp-basierte Programmiersprachen, wie Self und Java Script, Bestandteil der Studienarbeit. Als Machbarkeitsstudie soll ein Webportal für den Zugriff auf Patientenleitlinien, ähnlich den Webseiten der AWMF, auf Basis von Ercatons realisiert werden. Die Living Pages Research GmbH steht als Kooperationspartner zur Verfügung. Es werden einfache Anwendungsfälle wie Anmeldung, Navigation und Suche realisiert und anschließend auf die Vor- und Nachteile der Entwicklung dieses Portals mit Ercatons eingegangen.

2 Methodik

Die Vorgehensweise in dieser Studienarbeit gliedert sich in drei Abschnitte: Konzept, Programmierung und Technik, sowie Machbarkeitsstudie. Sie werden im Folgenden genauer beschrieben.

Konzept

Da Ercatons eine Variante des Prototyp-basierten Programmiersprachenansatzes darstellen, wurde zunächst die Prototyp-basierte Programmierung untersucht. Anschließend wurden auf Basis einer Literaturrecherche zu Prototyp-basierten Programmiersprachen die Sprachen Self und JavaScript näher betrachtet. Als nächster Schritt folgte die Betrachtung der Organischen Programmierung, wobei zunächst das grundlegende Konzept des Things recherchiert wurde. Anschließend wurden weitere Konzepte der Organischen Programmierung untersucht und zu den Prototyp-basierten Programmiersprachen abgegrenzt.

Programmierung und Technik

Die Programmierung und Technik wurde im darauf folgenden Schritt untersucht. Hierbei wurden die Ercatons dargestellt, eine Implementierung des Konzepts der Organischen Programmierung. Es wurden besonders die Spezifikation eines Ercatons, sowie die Ercato Engine und die verschiedenen Zugriffsarten betrachtet. Für den Software-Entwurf mit Hilfe von Ercatons wurden zunächst ausgewählte Konzepte der Ercatons-Programmierung, wie die Vererbung, mehrwertige Attribute und Aktionen, dargestellt. Anschließend folgte das Konzept zur Zugriffskontrolle, sowie ausgewählte Konzepte zur Persistenz, Abfrage und Anzeige.

Machbarkeitsstudie

Als Machbarkeitsstudie wurde ein Webportal mit Ercatons realisiert. Es wurde zunächst eine Anforderungsanalyse mit informeller Anforderungsbeschreibung und Funktionenbeschreibung erstellt. Hierauf folgte der Entwurf und anschließend die Implementierung des Portals. Eine Bewertung der Entwicklung dieses Webportals mit Hilfe von Ercatons und ein Epilog mit Zusammenfassung, zukünftigen Aufgaben und Resume runden die Studienarbeit ab.

3 Verwandte Arbeiten

Da die Organische Programmierung auf dem Konzept der Prototyp-basierten Programmierung basiert, stellt dieses Kapitel zunächst die Prototyp-basierte Programmierung vor. Anschließend wird eine kurze Einführung in verwandte Arbeiten gegeben. Hierzu werden die, ebenfalls Prototyp-basierten, Programmiersprachen *Self* und *JavaScript* erläutert.

3.1 Prototyp-basierte Programmierung

Prototyp-basierte Programmierung ist eine Variante der objektorientierten Programmierung (OOP), verwendet jedoch Prototypen als Alternative zum Klassenkonzept. Ein neues Objekt wird hierbei durch Kopieren anstatt durch Instantiieren erzeugt und es existiert, ebenso wie bei der Klassen-basierten Programmierung, ein Vererbungsmodell. Zur Beschreibung von gemeinsamen Eigenschaften verwandter Objekte werden in vielen objektorientierten Sprachen Klassen verwendet. Dies bietet allerdings einige Nachteile: Soll zum Beispiel ein Objekt erzeugt werden, so muss zunächst die Abstraktionsebene gewechselt und eine entsprechende Klasse definiert werden, von der anschließend das Objekt instantiiert und getestet wird. Prototypen hingegen erlauben es, in einer Abstraktionsebene zu bleiben und ein Objekt inkrementell und zur Laufzeit zu verändert. Es können Methoden und Attribute zu individuellen Objekten hinzugefügt und entfernt werden [Bor86]. Hierbei ist es auch möglich ein einziges Objekt mit einem eindeutigen Verhalten zu erstellen. Würde dies mit Hilfe des Klassen-Konzepts realisiert werden, so wäre für das Objekt eine eigene Klasse nötig und diese wiederum würde nur eine einzige Instanz besitzen. Klassen-basierte Sprachen wurden für Situationen entwickelt, in denen mehrere Objekte gleiches Verhalten besitzen. Dies ist bei Prototyp-basierten Sprachen nicht der Fall und es wird somit mehr Flexibilität gewährt.
Neue Objekte werden durch Kopieren von Prototypen erzeugt. Prototypen sind konkreter als Klassen, da sie Beispiele von Objekten darstellen anstatt nur eine Beschreibung des

Formats und der Initiierung, wie dies bei Klassen der Fall ist. Dies erhöht das Verständnis eines Programmcodes und erleichtert somit auch die Wiederverwendung von Modulen. Für den Programmierer ist es einfacher ein System anhand von Beispielen, als mittels einer reinen Beschreibung zu verstehen.
Ein weiterer Vorteil der Prototyp-basierten Programmierung besteht darin, dass die Beziehungen zwischen Objekten vereinfacht werden. Bei Klassen-basierten Sprachen existieren nach [US87] die beiden Beziehungen „is a“[1] und „kind of“[2], die für jedes Objekt bestimmt werden müssen. Anstelle dieser beiden Beziehungen wird beim Prototyp-Konzept nur eine „inherits from“ Beziehung benötigt, die angibt wie Objekte Verhalten und Zustand gemeinsam haben. Diese Vereinfachung erleichtert das Verständnis und Design von Vererbungshierarchien [US87].
Nicht jede Prototyp-basierte Sprache implementiert jedoch ein Vererbungskonzept, was den großen Nachteil hat, dass gemeinsames Verhalten nicht modelliert werden kann. Der Vorteil, den die Software-Welt gegenüber der realen Welt besitzt, nämlich das gleichzeitige Ändern mehrerer Verhaltensweisen, ist dann nicht mehr gegeben.
Zu den Prototyp-basierten Sprachen zählen unter anderem *Self* und *JavaScript*, auf die im Folgenden genauer eingegangen wird.

3.2 Self

Die Prototyp-basierte Programmiersprache Self wurde von David Ungar und Randall Smith entwickelt und ist das ursprüngliche Beispiel für das Prototyp-basierte Konzept. Sie basiert auf den einfachen drei Konstrukten: *Prototypen*, *Slots* (engl.: Steckplätze) und *Verhalten*. Es existieren weder Klassen noch Variablen und die Hauptbestandteile des Programmiermodells sind kommunizierende Objekte, sowie eine einfache Form der Vererbung [US87].
Self unterscheidet nicht zwischen Zustand und Verhalten eines Objekts. Es wird also kein Unterschied zwischen einem Variablen-Zugriff und einem Methodenaufruf gemacht.

1 Die „is a“ Beziehung in [US87] besagt, dass ein Objekt eine Instanz einer Klasse ist. Dies wird heutzutage üblicherweise als „instance-of“ Beziehung bezeichnet.
2 Die „kind of“ Beziehung in [US87] besagt, dass die Klasse eines Objekts eine Unterklasse eines anderen Objekts ist. Dies wird heutzutage üblicherweise als „is-a“ Beziehung bezeichnet, zum Beispiel in UML.

Objekte haben beliebig viele Slots. Diese besitzen einen Namen und können sowohl ein Attribut, als auch eine Methode beinhalten. Beides sind wiederum Objekte, die dem Aufrufer identisch erscheinen. Er weiss nicht, ob der angesprochene Slot eine Variable, eine Konstante oder eine Methode darstellt. Variablen-Objekte liefern sich selbst zurück während Methoden-Objekte Anweisungen ausführen und dessen Ergebnis zurückgeben [SU95].
Die fundamentale Operation in Self ist das Versenden von Nachrichten. Objekte greifen durch Nachrichten auf ihren Zustand zu. Sie senden diese an „self“, den Empfänger der Nachricht. Will zum Beispiel ein Punkt seinen „x“-Wert abfragen, so schickt er sich selbst eine „x“-Nachricht. Diese Nachricht findet den Slot mit dem Namen „x“ und führt das darin enthaltene Objekt aus. Da in diesem Fall der Slot nur eine Zahl beinhaltet, ist auch das Ergebnis der Abfrage diese Zahl. Soll ein Wert, zum Beispiel die Zahl „5“, in einem Slot gespeichert werden, so schickt sich der Punkt selbst eine Nachricht mit dem Wert „5“ an den Zuweisungs-Slot „x:“. Dieser enthält eine spezielle Methode, die den Wert im „x“ Slot abspeichert [SU95].
Da in Self der Variablen-Zugriff dem Methodenaufruf entspricht, können gewöhnliche Objekte als Methoden betrachtet werden, die sich selbst zurückliefern. Objekte sind also nicht mehr das was sie „sind“, sondern wie sie sich verhalten.
Erhält ein Objekt eine Nachricht für die es keinen passenden Slot beinhaltet, wird der Aufruf an ein Eltern-Objekt weitergegeben. Dies geschieht mit Hilfe des sogenannten *parent-Zeigers*. Auf diese Weise wird in Self Vererbung implementiert („inherits from“ Beziehung) und es ist somit möglich gemeinsames Verhalten von Objekten zu definieren. *Prototypen* kombinieren Vererbung und Instantiierung, was, im Gegensatz zu Klassen-basierten Programmiersprachen, die Flexibilität und Einfachheit erhöht. Neue Objekte werden durch Klonen (das heißt Kopieren) von Prototypen erzeugt. In Self kann jedes Objekt geklont werden [US87].

3.3 JavaScript

JavaScript, auch *JScript* oder *ECMA Script* genannt, ist eine weit verbreitete objekt-orientierte Skriptsprache und wird hauptsächlich dafür verwendet, Webseiten dynamisch zu gestalten [LR06]. Sie wurde von Brendan Eich bei Netscape entwickelt und ist durch

Self inspiriert [MT07]. Sie beinhaltet sowohl das Prototyp-Konzept, als auch das Konzept der Klassen und versucht auf diese Weise ein höheres und flexibleres Abstraktionslevel zu erreichen. Da die Wurzeln von JavaScript Prototyp-basiert sind, wird diese Programmiersprache im Folgenden untersucht.

3.3.1 Prototyp-Konzepte in JavaScript

JavaScript ist eine Prototyp-basierte Skriptsprache und besitzt eine zu Java und C/C++ ähnliche Syntax. Es existieren in Version 1.5, wie auch bei Self, keine Klassen und die Vererbung wird durch Prototypen realisiert [Hor01]. Jedes Objekt ist eine Sammlung aus Eigenschaften und kann zusätzlich spezielle, verborgene Eigenschaften besitzen. Zu diesen zählt zum Beispiel ein Link auf einen Prototypen (oder auf „null“ falls kein Prototyp für dieses Objekt existiert). Wird auf eine Eigenschaft des Objektes zugegriffen und kann diese im aktuellen Objekt nicht gefunden werden, so wird in dessen Prototyp danach gesucht. Wird einer Eigenschaft eines Objektes ein Wert zugewiesen, so ist der Prototyp davon nicht betroffen. Es wird nur das entsprechende Objekt geändert. Auf diese Weise ist es auch möglich Eigenschaften, die ein Objekt von Prototypen geerbt hat, zu überschreiben.
Ein neues Objekt wird durch Verwendung des „new“ Operators bei einem beliebigen Funktionsaufruf erzeugt (z.B. `new f(args)`). Bevor die Funktion aufgerufen wird, wird ein Objekt ohne Eigenschaften erzeugt und kann innerhalb dieser Funktion mit der „this“ Variable angesprochen werden. Die Funktion f selbst ist ein Objekt und besitzt bestimmte Eigenschaften. Zum Beispiel zeigt `f.prototype` auf denjenigen Prototypen, der für die Erzeugung des Objekts mit dem „new“ Aufruf verwendet wird. Eine Methode ist eine beliebige Funktion und kann sowohl an Objekte, als auch Prototypen angehängt werden. Die Methoden können mit Hilfe von „this“ auf das Objekt, von dem aus sie aufgerufen wurden, zugreifen.

3.3.2 Klassen-Konzepte in JavaScript

Mit JavaScript 2.0 [ECM09] wird die Version 1.5 um Klassen-basierte Objekte erweitert, die neben den Prototyp-basierten Objekten existieren. Eine Klasse wird definiert durch ein „class“-Statement, gefolgt von einem Namen und einem Block mit Eigenschaften und Funktionen dieser Klasse. Die Syntax um eine Eigenschaft aus einem Klassen-basierten

Objekt zu lesen oder es zu beschreiben unterscheidet sich nicht von Prototyp-basierten Objekten. Es wird jeweils die Form `object.property` verwendet. Klassen beinhalten eine Reihe von Anweisungen die zur gleichen Zeit ausgeführt werden und sie können von anderen Klassen erben, wobei Mehrfachvererbung nicht unterstützt ist.
Klassen wurden zu JavaScript hinzugefügt um ein höheres und flexibleres Abstraktionslevel als mit reinen Prototypen zu erreichen. Eine Vererbungshierarchie in JavaScript 1.5 mit Prototypen aufzubauen ist selbst für erfahrene Programmierer schwierig. Dies soll durch das Klassen-Konzept erleichtert werden, da die Klassen-Syntax selbsterklärender ist als eine Prototyp-Hierarchie [Hor01].
Die Verwendung zweier verschiedener Konzepte bietet jedoch nicht nur Vorteile. So ist es zum Beispiel nötig im Voraus zu entscheiden, ob für ein bestimmtes Problem Prototypen oder Klassen verwendet werden sollen, was nicht immer einfach ist und bei eventuellen Fehlentscheidungen zeitaufwändige Änderungen nach sich ziehen kann.

3.4 Zusammenfassung

Dieses Kapitel hat einen Überblick über das Konzept der Prototyp-basierten Programmierung gegeben und dabei die Prototyp-basierten Programmiersprachen *Self* und *JavaScript* näher erläutert. JavaScript vereint hierbei ab Version 2.0 sowohl das Konzept des Prototyps, als auch das Konzept der Klasse, während Self keine Klassen besitzt. Das vermittelte Wissen dient als Grundlage um das Verständnis der Organischen Programmierung zu erleichtern.

4 Die Idee der Organischen Programmierung

Dieses Kapitel beschreibt die Idee der Organischen Programmierung. Hierzu werden zunächst Idee und Eigenschaften eines Things erläutert um anschließend eine Einführung in die Organische Programmierung zu geben. Als Abschluß folgt eine Abgrenzung zur Prototyp-basierten Programmierung, welche im vorherigen Kapitel vorgestellt wurde.

4.1 Das Thing

Die Idee hinter einem *Ding* (Fachterm im Englischen: *Thing*) ist es, ein Objekt der realen Welt darzustellen ohne es im Voraus modellieren zu müssen. Stattdessen wird es nur beschrieben, dargestellt und verglichen. Oftmals werden erst während der Verwendung eines SW-Systems verschiedene Aspekte deutlich. Things können sich verändern und somit den aktuellen Wissensstand reflektieren. Sie benötigen kein Modell um existieren und sich weiterentwickeln zu können. Things haben sowohl Dokumenten- als auch Objekt-Charakter und besitzen die folgenden Eigenschaften [ILvW05]:
Sie sind eindeutig und es gibt niemals zwei identische Things. Dies ist auch in der realen Welt der Fall, denn jedes Lebewesen und jeder Gegenstand ist unterscheidbar, sei es durch einen kleinen Produktionsfehler. Ein Thing hat außerdem eine innere Struktur, welche sowohl Status, als auch Verhalten darstellt und getrennt von Algorithmen ist. Der Status wird durch einen Datenbaum, das Verhalten durch Reaktionen auf Nachrichten oder Ereignisse repräsentiert. Things besitzen Objekt-Charakter. Das bedeutet, dass ein Thing ein anderes Thing klonen oder von ihm erben kann (Vererbung). Außerdem können zwei Things unterschiedlich auf ein und dieselbe Nachricht reagieren (Polymorphismus) und ihre innere Struktur kann auf andere Things zeigen (Delegation). Zudem bestimmt ein Thing selbst wieviel seiner inneren Struktur sichtbar für andere Things ist (Kapselung).

Neben Objekt-Charakter besitzt ein Thing auch Dokumenten-Charakter. Es ist leicht lesbar und selbsterklärend, hat einen Besitzer, der die Sichtbarkeit und Kapselung kontrolliert und bestimmt seinen Lebenszyklus selbst. Persistenz und Versionisierung zählen ebenfalls zu den Eigenschaften des Dokumenten-Charakters. Status, Besitzer und Vererbungsbeziehungen eines Things können sich über dessen Lebenszyklus hinweg verändern. Dies wird auch als *Morphing* bezeichnet. Außerdem sind Things deterministisch, das heißt, ihre innere Struktur wird immer nur kurzzeitig inkonsistent sein. Sie sind somit transaktionssicher [ILvW04].
Eine weitere Eigenschaft ist die Projektion. Ein Thing definiert seine Interaktionen mit einem SW-System selbst. Es kann eine oder mehrere Repräsentationen bei der Manipulation durch Menschen (User Interface) oder Algorithmen (Programming Interface) haben. Ein Thing wird als *Zelle* (Fachterm im Englischen: *Cell*) [ILvW05] bezeichnet, wenn es zusätzlich zu den bisher genannten Eigenschaften eine eigene Aktivität oder Veränderung besitzt. Dieses Verhalten ist definiert durch Reaktion auf Ereignisse, wie zum Beispiel das Verstreichen einer bestimmten Zeitspanne. Der Vorgang des Entwickelns von Software mit Hilfe von Things wird *Thing-orientierte Programmierung* [ILvW04] genannt, wohingegen die *organische Programmierung* [ILvW05] das „Bauen“ und „Erweitern“ eines SW-Systems, das aus Things besteht und im Betrieb ist, meint.

4.2 Organische Programmierung

Der Begriff *organische Programmierung (OrP)* wurde von der Living Pages Research GmbH definiert und beschreibt ein Programmiermodell, welches den Prototyp-basierten Programmiersprachenansatz um einige wichtige Eigenschaften ergänzt und die Grenzen der traditionellen Software Entwicklung überschreitet [ILvW05].
Die Entwicklung von Programmiersprachen hatte schon lange zum Ziel, Dinge der realen Welt in Software-Objekte abzubilden, wobei objektorientierte Sprachen der aktuelle Stand der Entwicklung sind. Jedoch ist trotz jahrelanger Forschung der Unterschied zwischen diesen beiden Entitäten immer noch sehr groß und Software-Objekte zeigen Schwächen wenn es darum geht ein System zu “benutzen“ und zu „bauen“ anstatt es nur als Modell zu repräsentieren.
Das Prinzip der Model Driven Architecture (MDA) besteht zum Beispiel darin, ein

vollständiges Modell eines SW-Systems zu erstellen. Jedoch wird hierbei nicht beachtet, dass es oft unmöglich ist ein vollständiges und korrektes Modell eines Systems zu erstellen. Es spielen dabei nicht nur Zeitfaktoren, sondern auch Kosten und Fehlerfreiheit eine wichtige Rolle. Die Entwickler der OrP sind davon überzeugt, dass das Modellieren jedes einzelnen Details eines Systems gegen das generelle Prinzip der Software-Entwicklung, alles so einfach wie möglich zu halten, verstößt [ILvW04].
OrP ist keine neue Programmiersprache, sondern ein Modell, das auf dem Konzept des *Things* (vgl. Kapitel 4.1) basiert. Im Gegensatz zu Software-Objekten verhält sich ein Thing in einem SW-System wie ein selbständiges Objekt in der realen Welt. Ein SW-System wird nicht mehr komplett im Voraus modelliert, sondern *organisch*, das heißt Stück für Stück, aufgebaut und verändert.
In der klassischen Software-Entwicklung können Klassen zwar ebenfalls jederzeit geändert, neu kompiliert und getestet werden, bis die Software den gewünschten Anforderungen entspricht, dies kann aber besonders bei großen und komplexen Projekten zu Problemen führen. Jede Änderung erfordert das Überarbeiten des Entwurfs durch einen Programmierer oder Architekten. Dies verursacht sowohl hohe Kosten, als auch Einbußen an Flexibilität. Außerdem ist stets Expertenwissen für Änderungen nötig.
Komplexe SW-Systeme sind im Moment nicht objektorientiert, sondern bestehen aus Bausteinen die objektorientiert entwickelt wurden. Als Beispiel hierfür wird von den Entwicklern der Ercatons in [ILvW05] das World Wide Web (WWW) genannt. Es besteht aus Millionen unabhängig erzeugter Bestandteile. Dieses SW-System ist zu komplex um es immer wieder komplett neu zu kompilieren. Änderungen werden nur inkrementell und während der Laufzeit getätigt, zum Beispiel durch Hinzufügen von Text oder Bildern zu einer Webseite. Ein weiteres Beispiel sind Betriebssysteme, die aus Komponenten verschiedenster Quellen und Hersteller bestehen. Es werden immer wieder neue Bausteine hinzugefügt, die unmöglich im Voraus planbar gewesen wären.
Das Prinzip der OrP ist es, SW-Systeme zu entwickeln, die zwar immer noch objektorientiert sind, aber zusätzlich die Eigenschaft besitzen, sich spontan durch Konstruktion weiter zu entwickeln. In Abbildung 4.1 ist rechts die komplexe Software-Welt, die durch spontane Entwicklung entstanden ist, dargestellt. Auf der linken Seite befindet sich die weniger komplexe, rein objektorientierte Welt. Letztere besitzt als Kern das Objekt, während die rechte Seite hauptsächlich aus Dokumenten besteht und, wie zum Beispiel das WWW, nach und nach gewachsen ist. Zwischen diesen beiden Welten befindet sich

ein Übergang, wobei an einem Ende die Software geändert, neu kompiliert und getestet wird bis sie den Anforderungen entspricht, am anderen Ende die SW-Systeme für dieses Konzept bereits zu komplex sind. Das Ziel der OrP ist es, beide Seiten des Übergangs zu verbinden. Es beinhaltet somit sowohl das Konzept des Objekts, als auch das des Dokuments. Die Einheiten des Schnittpunkts zwischen den beiden Welten werden als Things bezeichnet [ILvW05].

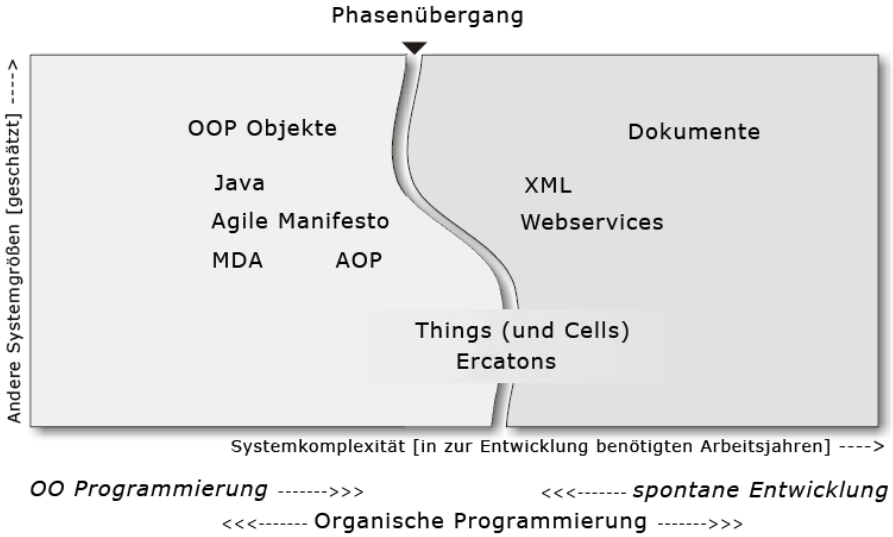


Bild 4.1: Grenzen von traditionellen objektorientierten Software-Systemen [ILvW05]

Die Philosophie hinter der OrP wird durch das Manifesto in Abbildung 4.2 zusammengefasst. Es beinhaltet sieben verschiedene Punkte, wobei die Erfinder des Manifestos in [ILvW05] behaupten, dass die objektorientierte Programmierung (OOP) nur die ersten drei Paragraphen („Die Ausnahme ist die Regel“, „Unsere Welt ist vielseitig und komplex anstatt wohl-strukturiert und einfach“ und „Software muss unregelmäßige, sich verändernde Muster anstatt regelmäßige Muster abdecken“) berücksichtigt. Die OrP soll nach [ILvW05] nicht nur diese, sondern auch die übrigen vier Punkte erfüllen. Sie besagen, dass ein SW-System ein organisches Wesen, anstatt eine Sammlung mathematischer Algorithmen darstellt und dass Software-Komponenten ein integraler Bestandteil unserer Welt und nicht Einheiten einer Meta-Ebene sind. Auch die beiden Paragraphen „Software entwickelt sich von klein zu groß anstatt von konkret zu abstrakt“ und „Software-Komplexität entsteht aus Einfachheit und Interaktion anstatt aus kompliziert-

tem Quellcode“ zählen zu denjenigen Punkten, die die OrP im Gegensatz zur OOP erfüllt.

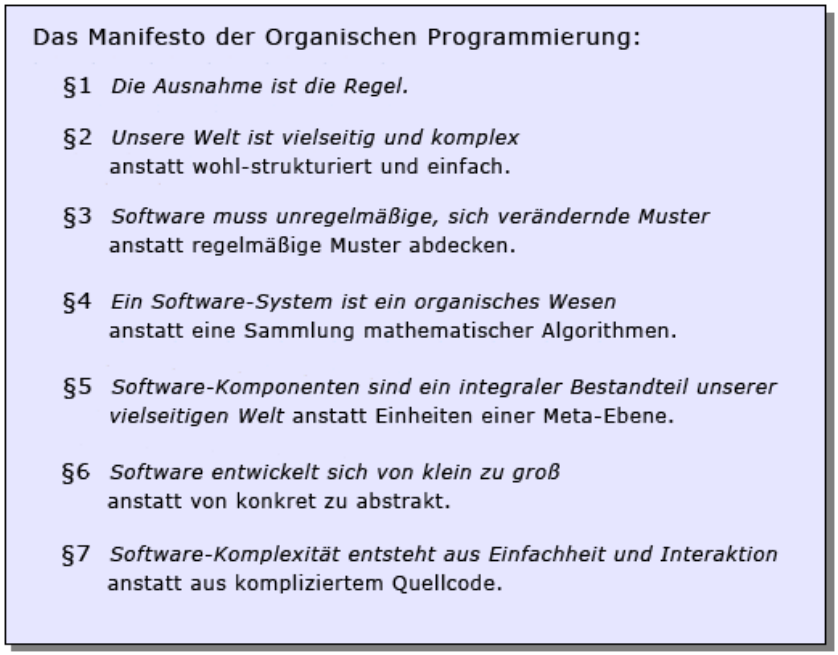


Bild 4.2: Manifesto der Organischen Programmierung [ILvW05]

4.3 Abgrenzung der Organischen Programmierung zur Prototyp-basierten Programmierung

Das Ziel der OrP ist es, das reale Universum möglichst verlustarm auf das Universum von Maschinen abzubilden. Hierbei wurde ein Konsens zwischen formalen und informellen Abbildungen gefunden. Prototyp-basierten Programmiersprachen wie Self oder JavaScript fehlen einige wichtige Eigenschaften, die im Folgenden näher erläutert werden:
Things sind, im Gegensatz zu Programmiersprachenobjekten, robust und bleiben trotz zufälliger Änderungen gültige Things. Zufällige Änderungen sind meist Fehler, die zwar ein Thing nicht unbedingt verbessern, aber zum Beispiel zwei Things zu einem neuen verschmelzen können. Things sind also geeigneter als Programmiersprachen um organische, sich verändernde oder entwickelnde Strukturen darzustellen.
So wie in der realen Welt Dinge sichtbar und greifbar sind, besitzen auch Things ein Aussehen und bestimmte Eigenschaften, durch die sie beschrieben werden. Sie haben

Dokumenten-Charakter und eine eigene Präsentationsschicht.
Things sammeln, wie dies auch bei Prototyp-basierten Programmiersprachen der Fall ist, mit der Zeit Informationen und entwickeln sich auf diese Weise weiter. Sie lassen sich also zur Laufzeit beliebig verändern. Der Unterschied zu Sprachen wie Self oder JavaScript ist hierbei allerdings, dass deren Objekte einen kürzeren Lebenszyklus haben. Sie bilden eher einzelne technische Aspekt ab, anstatt eine eigenständige Entität der realen Welt. Things sind, im Gegensatz zu Self- oder JavaScript-Objekten, von Grund auf persistent. Zum Beispiel beinhalten Self-Objekte als Realisierung ihrer Slots nur Zeigeradressen. Diese verlieren jedoch bei der Persistierung des einzelnen Objektes an Bedeutung. Wird hingegen der Objektgraph persistiert, so führt dies dazu, dass ein Snapshot der gesamten virtuellen Maschine gemacht werden muss. Ein einzelnes Self-Objekt ist somit auch nicht von einer virtuellen Maschine in eine andere transferierbar, wie dies bei Things der Fall ist. Das Things-Konzept erlaubt es, einzelne Things aus der Engine zu entnehmen, zu verändern und wieder einzubinden.
Objektorientierte Programmiersprachen besitzen also keine echten wiederverwendbaren Objekte. Sie sind immer auf eine bestimmte Anwendung oder Ausführungszeit beschränkt, es sei denn, sie bilden nur ein mathematisches Konzept direkt ab. Things hingegen „leben“ in einem Thing-Universum, sie sind eigenständig persistent, wiederverwendbar, können wachsen und sich weiter entwickeln.

4.4 Zusammenfassung

In diesem Kapitel zur OrP wurde zunächst das Konzept des Things erläutert. Hierbei wurde besonders betont, dass ein Thing sowohl Dokumenten- als auch Objekt-Charakter besitzt. Anschließend wurden die Eigenschaften eines Things, wie zum Beispiel Polymorphismus, Kapselung und Delegation, aufgezeigt. Aufbauend auf dem Konzept des Things wurde die Idee der Organischen Programmierung beschrieben und anschließend zur Prototyp-basierten Programmierung abgegrenzt.

5 Ercatons als technische Infrastruktur

Ercatons stellen eine Implementierung des Konzepts der Organischen Programmierung dar. Dieses Kapitel bietet einen Überblick über die Grundlagen der Ercatons. Zunächst wird ein Einblick in die Spezifikation eines Ercatons, die Ercato Markup und die verschiedenen Ercaton Typen gewährt. Anschließend wird der Aufbau der Ercato Engine beschrieben. Es exisitieren verschiedene Möglichkeiten um auf Ercatons zuzugreifen, wie zum Beispiel Web-Zugriff, Remote-Zugriff mit Hilfe der sogenannten Ercato Shell und der Zugriff über eine Programmiersprache. Diese werden als Abschluss des Kapitels vorgestellt.

5.1 Das Ercaton

Der Begriff *Ercaton* ist zusammengesetzt aus *(M)ercato* und *(I)on*, was soviel bedeutet wie „elementares Markt-Teilchen“ [ILvW05]. Ercatons stellen eine Implementierung des OrP Programmiermodells dar. Sie trennen innere Struktur von Algorithmen, wobei die innere Struktur durch XML und Algorithmen in einer Programmiersprache, wie zum Beispiel Java oder XSLT repräsentiert werden. Im folgenden Abschnitt wird die Spezifikation eines Ercatons beschrieben und anschließend auf die Syntax, die sogenannte *Ercato Markup*, eingegangen. Neben einer Auflistung der verschiedenen Typen, die ein Ercaton annehmen kann, wird auch das Ercato Programmiermodell beschrieben.

5.1.1 Spezifikation eines Ercatons

Ercatons sind *Cells* (siehe Kapitel 4.1) mit den folgenden Spezifikationen [ILvW05]: Ein Ercaton besitzt einen eindeutigen lokalen Namen nach Unix Shell Konventionen. Er ist von der Form `~Besitzer/Pfad[,version]`. Die *Ercato Markup*, die im Kapitel 5.1.2 dargestellt wird, ist die Syntax eines Ercatons. Sie besteht aus XML-Elementen

und Attributen und existiert in einem separaten Namensraum. Durch die Ercato Markup werden alle semantischen Eigenschaften eines Things beschrieben. Es gibt sechs verschiedene Typen von Ercatons: *plain*, *role*, *prototype*, *resource*, *index* und *version*. Sie werden im Kapitel 5.1.3 genauer beschrieben. Ein Ercaton kann zwar seinen Typ ändern, jedoch niemals gleichzeitig mehrere Typen annehmen. Mithilfe einer speziellen Algebra, dem sogenannten *XOperator*, können Ercatons addiert und subtrahiert werden und auf diese Weise von anderen Ercatons erben (siehe Kapitel 6.1.1). Ercatons besitzen zudem ein bestimmtes Verhalten. Sie können *actions*, *trigger*, *targets* und *objects* beinhalten. Eine *action* legt das Verhalten nach Erhalt einer Nachricht fest. Actions werden durch Zugriffsrechte geschützt und können Ercatons als Parameter erhalten und zurückgeben. *Trigger* bestimmen die Reaktion auf Ereignisse, wie zum Beispiel nach Ablauf einer bestimmten Zeitspanne oder Erhalt einer Email. *Targets* sind ein Mechanismus zur Beeinflussung der Benutzeroberfläche und *objects* stellen die Projektion auf eine API für einen Algorithmus dar. Das Verhalten kann durch einen Algorithmus implementiert und die Durchführung an andere Ercatons delegiert werden. Ein Ercaton schützt seine innere Struktur und Aktionen durch Rollen-basierte Zugriffsrechte. Kapitel 6.2 beschreibt das Zugriffsmodell detailliert. Es existieren die Status lesbar (r), schreibbar (w), navigierbar (b), ausführbar (x), „tausche Rolle durch Besitzer der Aktion“ (s) und „Aktion ist ausführbar und kann das aktuelle Ercaton verändern“ (t). Mit Zugriffsrechten können sowohl Rollen-basierte Geschäfts-Logik, als auch die Paket-basierte objektorientierte Kapselung realisiert werden. Eine weitere Eigenschaft von Ercatons ist die Distribution. Sie können verteilt werden, das bedeutet, dass der Zugriff auf Aktionen nicht an den Adressraum einer Maschine gebunden ist. Die Übertragungsprotokolle hierfür sind Hypertext Transfer Protocol (HTTP) und Simple Object Access Protocol (SOAP).

5.1.2 Ercato Markup

Die *Ercato Markup* stellt die Syntax eines Ercatons dar [ILvW05]. Sie beinhaltet XML-Elemente und Attribute und kann durch eine Namensraumdefinition in jedes beliebige XML-Dokument eingefügt werden. Folgende Definitionen stehen zur Verfügung: Der *erc: Namensraum* lautet „http://ercato.com/xmlns/ErcatoCore“. Er definiert Elemente und Attribute des Kerns der *Ercato Machine*. Neben diesem existieren zwei weitere

Namensräume, der *erx: Namensraum* „http://ercato.com/xmlns/ErcatoExtensions“ und der *e: Namensraum* „http://ercato.com/xmlns/ErcatoDefaultPattern“. Sie beinhalten erweiterte Elemente.
Das Codebeispiel 5.1 zeigt ein einfaches „Hello World“- Ercaton. Hierbei stellt `~demouser` den Besitzer und `hello` den Namen des Ercatons dar. Beide Teile formen zusammen die eindeutige *Ercaton-ID*. Um das Ercaton anschließend zu betrachten genügt der Aufruf von `~demouser/hello` im Browser.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<hello xmlns:erc="http://ercato.com/xmlns/ErcatoCore">
  <erc:id>~demouser/hello</erc:id>
  <p>Hello World!</p>
</hello>
```

Quellcode 5.1: Hello World - Ercaton

Die folgenden Codebeispiele werden aus Gründen der Übersichtlichkeit ohne die XML-Definition `<?xml version="1.0" encoding="ISO-8859-1"?>` und die XML-Namensraumdefinitionen dargestellt.

5.1.3 Ercaton Typen

Es gibt sechs verschiedene Typen von Ercatons [ILvW05]. Ein Ercaton kann seinen Typ über den Lebenszyklus hinweg wechseln, aber nie mehr als einen zur selben Zeit annehmen.

5.1.3.1 Plain

Das einfachste Ercaton ist vom Typ *plain*. Dieser Typ ist Standard für jedes Ercaton, solange kein anderer Typ spezifiziert ist. Er muss nicht explizit definiert werden.

5.1.3.2 Role

Ein Role-Ercaton kann ein Benutzer, ein Besitzer, eine Gruppe oder eine Rolle sein. Der Typ *user* ist ein Untertyp von *role*. Er muss authentifiziert werden und stellt entweder einen Menschen (Benutzer, Besitzer), eine Gruppe oder eine geschützte

Domäne, wie zum Beispiel ein Software-Paket, dar. Das Codebeispiel 5.2 zeigt ein User-Ercaton für den Benutzer Max Mustermann. Der Benutzer `~admin` erhält hier mit `<erc:permission role="~admin">rwb</erc:permission>` das Recht dieses User-Ercaton zu lesen und zu editieren. Auf die Rollen- und Benutzer-Verwaltung wird in Kapitel 6.2 detailliert eingegangen.

```
<user>
  <erc:id>~max</erc:id>
  <erc:type>user</erc:type>
  <erc:permission role="~admin">rwb</erc:permission>
  <person>
    <first-name>Max</first-name>
    <last-name>Mustermann</last-name>
  </person>
</user>
```

Quellcode 5.2: User-Ercaton für Max Mustermann

5.1.3.3 Prototype

Der Typ *prototype* definiert einen Prototypen, von dem neue Ercatons erzeugt werden können. Diese Ercatons enthalten dann alle Eigenschaften des Prototypen und können weiter verändert werden. Wird der Prototyp nachträglich verändert, so werden diese Änderungen auch an die von ihm abgeleiteten Ercatons weitergegeben. Codebeispiel 5.3 zeigt ein Prototyp-Ercaton für eine Kundenadresse. In Abschnitt 6.1.1 werden die Vererbungskonzepte der Ercatons genauer beschrieben.

5.1.3.4 Index

Ein Ercaton vom Typ *index* stellt ein formales Datenschema oder andere formale Informationen dar. Codebeispiel 5.4 zeigt ein Index-Ercaton mit zwei Indizes für Kundenadressen. Es werden der Kundenname, sowie dessen Wohnort indiziert. Mit Hilfe von Index-Ercatons können Abfragen erstellt werden, wie zum Beispiel „Welche Kunden sind in Erlangen wohnhaft?“.

```
<address>
  <erc:id>~sample/address</erc:id>
  <erc:type>prototype</erc:type>
  <customer>
    <name erx:field-name="Name"/>
    <street erx:field-name="Straße"/>
    <city erx:field-name="Wohnort"/>
    <country erx:field-name="Land"/>
  </customer>
</address>
```

Quellcode 5.3: Prototyp-Ercaton für eine Kundenadresse

```
<customers>
  <erc:type>index</erc:type>
  <erc:id>~sample/idx/customers</erc:id>
  <erc:index>
    <erc:name>name</erc:name>
  </erc:index>
  <erc:index>
    <erc:name>city</erc:name>
  </erc:index>
</customers>
```

Quellcode 5.4: Ein Index-Ercaton für Kundenadressen

5.1.3.5 Resource

Binäre Daten, wie zum Beispiel ein pdf-Dokument, ein Film oder ein Code Archiv, werden als *Resource-Ercaton* bezeichnet und erhalten von der Engine automatisch den Typ *resource*. Über eine Referenz können sie in gewöhnliche Ercatons eingebunden werden. Hier werden dann auch wichtige Informationen, wie Autor, Editierfelder oder ein spezielles View Target gespeichert. Resource-Ercatons selbst beinhalten weder Aktionen, noch Targets oder Index-Einträge.

5.1.3.6 Version

In einem *Version*-Ercaton wird die Versionshistorie eines Ercatons gespeichert. Ein Version-Ercaton kann wie ein gewöhnliches Ercaton behandelt werden und wird über

die reservierte Version „v“ angesprochen. Um ein Ercaton zu versionisieren genügt das Einfügen eines `<version />`-Tags in das entsprechende Ercaton. Das zugehörige Version-Ercaton wird dann von der Engine selbständig angelegt, verwaltet und gelöscht. Die Engine kontrolliert zudem manuelle Änderungen an einem Version-Ercaton und erzwingt ein Transaktions-Rollback, falls ungültige Angaben erfolgen. Außerdem kann ein Version-Ercaton Callback-Aktionen definieren, die das Hochzählen von Versionen steuern. Wird hierbei die aktuelle Version gewählt, so wird keine neue Version angelegt, sondern die aktuelle direkt geändert. Daher wird zu jeder Version sowohl ein Anlege-, als auch ein Änderungsdatum gespeichert. Zudem enthalten Version-Ercatons Aktionen um eine alte Version zu aktivieren oder sie zu löschen. Codebespiel 5.5 zeigt ein Version-Ercaton für eine Preisliste. Nach jeder Preisänderung wird eine neue Version des Ercatons erzeugt und dies im Version-Ercaton mit Datum und Link auf das Ercaton vermerkt. Das Version-Ercaton enthält nun die komplette Versionshistorie und kann per `~sample/shop/price1,v` angesprochen werden. Auf die jeweiligen Versionen des versionisierten Ercatons wird mit Hilfe von `~sample/shop/price1,1`, `~sample/shop/price1,2` u.s.w. zugegriffen, während mit `~sample/shop/price1` immer automatisch das aktuelle Ercaton ausgewählt wird.

```
<history>
  <erc:type>version</erc:type>
  <erc:id>~sample/shop/price1</erc:id>
  <erc:version>v</erc:version>
  <erc:action name="get-active"></erc:action>
  <erc:action name="increment"></erc:action>
  <erc:action name="activate"></erc:action>
  <erc:action name="purge"></erc:action>
  <erc:versions>
    <erc:version>
      <erc:id-ref>~test/shop/price1,1</erc:id-ref>
      <erc:active>true</erc:active>
      <erc:created-at>22.01.2010 11:51:48</erc:created-at>
      <erc:created-by>~sample</erc:created-by>
      <erc:reason>created</erc:reason>
    </erc:version>
    <erc:version>
```

```
18        <erc:id-ref>~test/shop/price1,v</erc:id-ref>
19        <erc:created-at>22.01.2010 11:51:48</erc:created-at>
20        <erc:created-by>~sample</erc:created-by>
21        <erc:modified-as>version</erc:modified-as>
22        <erc:reason>changed</erc:reason>
23      </erc:version>
24    </erc:versions>
25  </history>
```

Quellcode 5.5: Ein Version-Ercaton für eine Preisliste

5.1.4 Ercato Programmiermodell

Ercatons sind Programmiersprachen-unabhängig [ILvW05]. Das *Ercato Programmiermodell* konzentriert sich auf das Erzeugen, Ändern und Benutzen von Ercatons. Letzteres bedeutet, dass Aktionen ausgeführt und der Status eines Ercatons ausgelesen werden kann. Außerdem können sie dem Benutzer als Teil der Benutzeroberfläche angezeigt oder an ein anderes System gesendet werden.
Es werden drei Fälle bei der Implementierung von Aktionen unterschieden:

1. Administrative Aufgaben, wie zum Beispiel das Ändern eines Status. Diese Aufgaben werden an *Service-Ercatons* (siehe 5.2.2) delegiert.
2. Definition der Benutzeroberfläche. Dies wird von Service-Ercatons, die XSLT-Stylesheets sind, implementiert.
3. Algorithmische, nicht-triviale Aufgaben. Sie werden mithilfe einer objektorientierten Programmiersprache erzeugt.

Momentan unterstützt die Ercato Engine nur die Sprache Java, Support für weitere Sprachen kann jedoch hinzugefügt werden. Die Implementierungssprache von Aktionen ist verborgen. Aktionen können sich gegenseitig aufrufen, auch wenn sie in verschiedenen Sprachen implementiert wurden. In Abschnitt 6.1.4 wird die Implementierung einer Aktion mit Hilfe von Java genauer beschrieben.

5.2 Ercato Engine

Um mit Ercatons arbeiten zu können ist neben der Spezifikation eines Things eine virtuelle Maschine nötig. Die sogenannte *Ercato Engine* implementiert die vorgestellten Ideen des OrP Programmiermodells. Im folgenden wird die ErcatoJ Engine, eine Ercato Engine die auf Java Enterprise Edition (Java EE) basiert, beschrieben und im Anschluss auf die *Erxlib* eingegangen. Sie ist eine Erweiterung des Ercato Kernels und unterstützt ihn mit einer Menge an Standard-Funktionalitäten. Eine Beschreibung der sogenannten *Builtin-Ercatons* rundet diesen Abschnitt ab.

5.2.1 ErcatoJ Engine

Die ErcatoJ Engine wurde von der Living Pages Research GmbH entwickelt [Liv09a]. Sie stellt im Gegensatz zu Frameworks ein eigenes Programmiermodell zu Verfügung und besteht aus einer Menge an Enterprise JavaBeans (EJBs), die bei Änderungen an Ercatons kein Redeployment benötigen. Diese EJBs integrieren sich in eine existierende Java EE Anwendung. Jedes Ercaton wird durch ein Enterprise JavaBean (EJB) repräsentiert. Abbildung 5.1 zeigt, dass die ErcatoJ Engine komplett auf dem Java EE Anwendungsserver basiert und mit anderen Java EE Anwendungen interagieren kann. Dem Java EE Anwendungsserver liegt ein Betriebssystem zusammen mit einer Datenbank zugrunde. Die Datenbank wird nur für Suchabfragen und nicht zum Persistieren verwendet. Es ist nicht möglich direkt auf die Datenbank zuzugreifen, sie dient lediglich der Beschleunigung von Abfrage-Operationen. Das Konzept der Ercatons stellt sicher, dass die Datenbankinhalte keinerlei Informationen abbilden, die sich nicht auch in Ercatons finden.

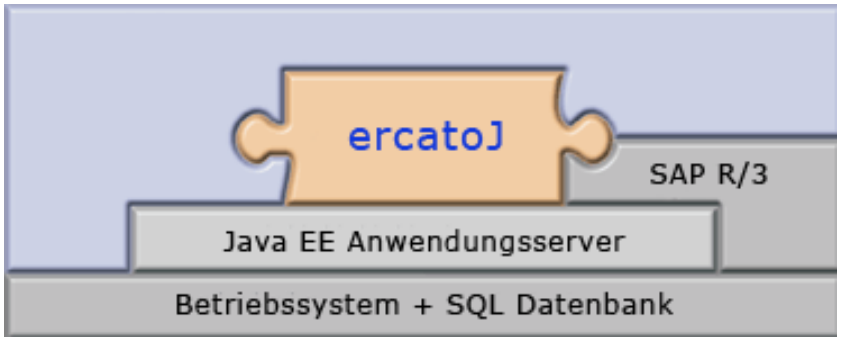


Bild 5.1: Die ErcatoJ Engine [Liv09a]

5.2.2 Erxlib

Wird die Ercato Engine mit einem Betriebssystem verglichen, so stellt sie den Kernel der Ercato-Welt dar [ILvW04]. Die Ercato Spezifikation besagt, dass nur Funktionalitäten, welche nicht im Benutzerraum implementiert werden können, Teil des Kernels sind [IL09]. Der Benutzerraum hingegen ist alles, was durch andere Ercatons implementiert werden kann. Diese Architektur ist ähnlich einem kleinen UNIX Kernel. Der Kernel wird durch eine Menge an Standard-Funktionalitäten ergänzt, den Binärdateien unter `/usr/bin`. Auch der Ercato Kernel wird durch Standard-Funktionalitäten erweitert. Diese werden in Form von Service-Ercatons durch die sogenannte *Erxlib* (Kurzform für *Ercato Extension Library*) zur Verfügung gestellt. Ihr Funktionsumfang ist mächtiger als der, den die Ercato Engine ursprünglich beinhaltet und es ist möglich die Erxlib beliebig zu erweitern. In Abbildung 5.2 sind auf oberster Ebene Business-Ercatons, mittig die Erxlib und auf unterster Ebene der Ercato Kernel dargestellt.

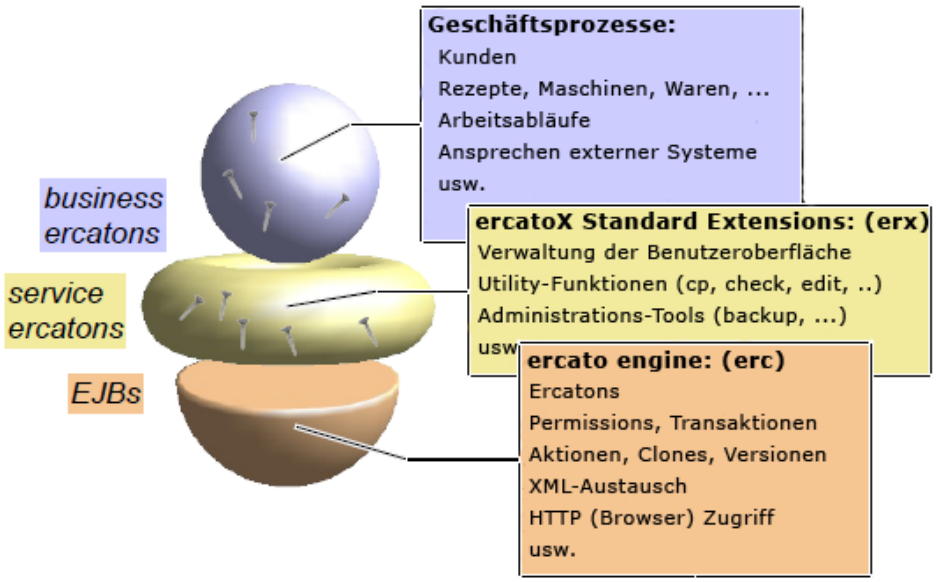


Bild 5.2: Das erxlib Layer [IL09]

Der Kernel besteht aus Ercatons und implementiert zum Beispiel Aktionen, Permissions, Transaktionen und Versionisierung. *Business-Ercatons* implementieren Gegenstände der realen Welt, wie zum Beispiel Kunden, Rezepte, Maschinen oder Waren. Die sogenannten *Service-Ercatons* hingegen sind Teil der Erxlib und stellen Utility-Funktionen

zur Verfügung. Typische Programmiersprachen-Konzepte sind Kopieren, Editieren, Ändern des Status und Abfragen von Ercatons. Auch das Verwalten der Benutzeroberfläche ist Aufgabe der Service-Ercatons. Neben vielen anderen existieren die folgenden Service-Ercatons: *edit*, *cp*, *echo* und *catalog*. Ihre ID ist von der Form `/bin/...` und sie sind dem Benutzer `~root` zugeordnet. Es werden also Root-Rechte benötigt um sie zu editieren. Das Codebeispiel 5.6 ruft im `/bin/cp` Ercaton die Aktion *copyAndEdit* auf, um das aktuelle Ercaton zu kopieren und anschließend zu editieren.

```
1 <erc:action name="copy" erx:field-name="Neue Adresse anlegen">
2     /bin/cp!copyAndEdit
3 </erc:action>
```

Quellcode 5.6: Aufruf des Service Ercatons 'cp'

5.2.3 Builtin-Ercatons

Builtin-Ercatons sind Teil der Ercato Engine und müssen nach Installation der Engine ebenfalls installiert werden. Im Gegensatz zu Service-Ercatons dürfen Builtin-Ercatons nicht fehlen oder defekt sein. Sie sind nicht Bestandteil der Erxlib, können jedoch durch neue Versionen aus der Erxlib überschrieben werden. Ihre ID ist oftmals von der Form `/builtin/...`. Das Builtin-Ercaton `/builtin/lscatalog`, zum Beispiel, ist ein Index-Ercaton, welches die IDs aller Ercatons indiziert. Auch Resource-Ercatons, die ansonsten in keinem Index auftauchen, werden hierbei berücksichtigt. Es ist dann möglich Abfragen über die Gesamtzahl aller Ercatons auszuführen.
Es existieren aber auch Builtin-Ercatons, die nicht über eine Namenskonvention verfügen. Ein Beispiel hierfür sind die beiden Role-Ercatons `~any` und `~anonymous`.
Ein Builtin-Ercaton, das später häufig Verwendung findet, ist `/builtin/adder`. Dieses Ercaton ist ein reservierter Platzhalter im Namensraum aller Ercatons und sammelt Informationen, die per `erc:add` Attribut indiziert wurden. Es wird zum Beispiel zur Implementierung von Bäumen mit Referenzzählern verwendet und in Abschnitt 6.1.3 genauer erklärt.

5.3 Zugriff auf Ercatons

Die wichtigsten Zugriffsarten auf Ercatons sind *Web Access*, *Remote Access* und der Zugriff über eine Programmiersprache [IL09]. Sie werden im Folgenden beschrieben. Außerdem wird die sogenannte *XReferenz* erläutert. Mit deren Hilfe ist es möglich, die XML-Repräsentation oder grafische Oberfläche von Ercatons anzuzeigen, sowie Aktionen auszuführen und Parameter zu übergeben.

5.3.1 XReferenz

Eine *XReferenz* ist ein Ercaton-Querverweis, mit dessen Hilfe es möglich ist die XML-Repräsentation eines Ercatons, oder dessen grafische Oberfläche anzeigen zu lassen. Außerdem können Aktionen aufgerufen und ihnen Parameter übergeben werden. Die XReferenz besteht aus der *Ercaton-ID*, einem Target, welches mit `:target` angehängt wird, und/oder einem Aktionsnamen, der mit `!action` spezifiziert wird. Parameter können in Klammern übergeben werden. Die Beispiele in 5.7 zeigen XReferenzen beim Aufruf im Browser. Hierbei wird mit `:null` die XML-Repräsentation eines Ercatons angezeigt, während `:view` die grafische Anzeige des Ercatons aufruft. Für den Webzugriff ist `:view` der Standardwert und kann deshalb auch weggelassen werden. Beim Zugriff über die Ercato Shell (ESH) (vgl. 5.3.3) ist der Standardwert `:null`. Mit `!action` wird eine Aktion aufgerufen (hier zum Beispiel die Aktion *main*) und `(amount=1)` gibt einen Argumentnamen mit Standardwert an. Die detaillierte Struktur der XReferenz befindet sich im Anhang unter A.2.

```
1 ~demo/counts/count
2 ~demo/counts/count:view
3 ~demo/counts/count:null
4 ~demo/counts/count!main
5 ~demo/counts/count!main:null
6 ~demo/counts/count!main(amount=1)
```

Quellcode 5.7: XReferenz Beispiele

5.3.2 Web-Zugriff

Web-Zugriff auf ein Ercaton wird durch Aufruf der URL

`http[s]://<engine_host>[:<engine_port>]/erc/aton/<XReference>`

im Browser realisiert. Der Port der Engine (`<engine_port>`) lautet 80+`<engine_number>` für HTTP und 443+`<engine_number>` für HTTPS. Als Standard besitzt die Engine die Nummer 0. Die XReferenz (`<XReference>`) ist ein Ercaton-Querverweis und wurde in Abschnitt 5.3.1 genauer erklärt.
Es ist anstelle des URL-Musters `/erc/aton/` auch möglich über `/erc/atons/` auf das Ercaton zuzugreifen. Hierbei erzwingt `/erc/atons/` einen Login, während `/erc/aton/` einen „Permission-Denied“-Fehler wirft, sobald ein Ercaton, welches nicht durch entsprechende Rechtevergabe den Zugriff auch ohne Login gestattet, aufgerufen wird.

5.3.3 Remote-Zugriff mit Hilfe der Ercato Shell

Die ESH ist ein *Command Line Tool* und ein Application Programming Interface (API) zur Interaktion mit der Ercato Engine. Sie führt Befehle aus, indem sie sie über ein Ercato-HTTP-Protokoll an den Ercato Server schickt.
Durch Eingabe von

```
$ esh
```

wird eine Hilfe zu den ESH-Befehlen mit Verwendung und Beispielen aufgerufen. Codebeispiel 5.8 zeigt einen Ausschnitt dieser Hilfe mit den wichtigsten Optionen. Die ESH besitzt die eingebauten Befehle *put*, *get*, *ls*, *rm*, *exec*, *login* und *logout*, wobei *ls* momentan noch nicht implementiert ist.

Anmeldung und Abmelden an der ESH

Durch Verwendung von ESH-Befehlen kann auf Ercatons remote zugegriffen werden. Es ist zunächst nötig sich an der ESH mit Hilfe des *login*-Befehls anzumelden:

```
$ esh login -s mysession -h <engine_host>[:<engine_port>] username
```

```
usage: esh put|get|ls|rm|exec|login|logout
           -r     (raw/resource)
           -v     (verbose)
           -o     (override, in put)
           -a     (all, in rm)
           -f/-ff <localfile>
           -h     <host[:port]>
           -e     <encoding>
           -p     <permission>
           -s     <sessionname>
           <id>
         [<parameter>=<value>]
```

Quellcode 5.8: Verwendung des esh-Befehls

Wie beim Web-Zugriff müssen auch hier der Port (`<engine_port>`) und Host (`<engine_host>`) der Engine angegeben werden. Dies wird von der ESH als Session „mysession“ gespeichert. Anschließend kann, zum Beispiel auf das Service-Ercaton `/bin/ls`, mit folgendem Befehl zugegriffen werden:

```
$ esh /bin/ls -s mysession
```

Wird keine Session angegeben, so wird eine Defaultsession verwendet. Der „-s“ Parameter erlaubt es, mehrere Engines/User parallel mit der ESH zu nutzen. Mittels des *logout*-Befehls ist es möglich sich von der ESH abzumelden.

Einfügen und Abfragen eines Ercatons

Um ein Ercaton in die Ercato Engine einzufügen, steht der ESH-Befehl *put* zur Verfügung:

```
$ esh put -f <Dateiname>
```

Die Option „-f“ gibt hierbei an, dass es sich um eine lokale Datei handelt und `<Dateiname>` spezifiziert den Namen der Datei. Es ist möglich, sowohl eine *.txt*, als auch eine *.erc* Datei in die Engine zu laden. Die Datei muss allerdings eine gültige Ercaton-ID beinhalten. Der *put*-Befehl benötigt keinen weiteren Parameter um den Ort, an den das Ercaton geladen werden soll, zu spezifizieren. Dies ergibt sich aus der Ercaton-ID und den Parametern der aktuellen Session.
Mit Hilfe des *get*-Befehls kann das Ercaton mit der entsprechenden ID aus der Engine abgefragt werden:

```
$ esh get <Ercaton-ID>
```

Ausführen einer Aktion

Das Ausführen einer Aktion erfolgt durch den *exec*-Befehl:

```
$ esh exec <XReference>
```

Ein Beispiel für eine XReferenz wäre:

```
~demo/counts/count!main(amount=1)
```

Löschen eines Ercatons

Ein Ercaton kann mit Hilfe des folgenden *rm*-Befehls gelöscht werden:

```
$ esh rm [-a] <Ercaton-ID>
```

Die Option `-a` (all) wird verwendet um auch versionierte Ercatons vollständig zu löschen. Ansonsten würden sie nur deaktiviert werden.

5.3.4 Zugriff über eine Programmiersprache

Neben Web- und Remote Access is auch der Zugriff auf Ercatons über eine Programmiersprache, wie zum Beispiel Java, möglich. Ercatons erscheinen unter Java als *com.ercato.core.ErcatonObject* oder eine Unterklasse hiervon. Es ist möglich entweder über die API dieses Objektes zu kommunizieren, oder direkt über die XML Repräsentation des Ercatons, welches ein *org.w3c.dom.Document* Objekt ist. In Abschnitt 6.1.4 wird die Implementierung einer Aktion mit Hilfe von Java beschrieben.

5.4 Zusammenfassung

Dieses Kapitel hat einen Überblick über die Grundlagen der Ercatons gegeben. Nach einer Einführung in die Spezifikation eines Ercatons wurde dessen Syntax, sowie die verschiedenen Ercaton Typen genauer beschrieben. Auch das Ercato Programmiermodell wurde erläutert. Eine Beschreibung der Ercato Engine diente dem Verständnis des Entwickeln mit Ercatons und gewährte einen Einblick in die Erxlib. Abschließend wurden die verschiedenen Zugriffsarten auf Ercatons beschrieben und gezeigt, wie es möglich

ist Ercatons in die Engine einzufügen, sie abzurufen und zu löschen. Dies bildet eine Wissensgrundlage für das folgende Kapitel zum Software-Entwurf mit Hilfe von Ercatons, sowie für die Entwicklung des Webportals für Patientenleitlinien.

6 Software-Entwurf mit Hilfe von Ercatons

In diesem Kapitel wird der Software-Entwurf mit Hilfe von Ercatons vorgestellt. Zunächst werden ausgewählte Konzepte der Ercatons-Programmierung, wie die Vererbung, mehrwertige Attribute und die Implementierung von Aktionen mit Hilfe von Java erläutert. Anschließend wird das Konzept zur Zugriffskontrolle aufgezeigt, welches mit Hilfe von rollenbasierten Zugriffsrechten Status und Aktionen von Ercatons schützt. Der interne Persistenzmechanismus, Index und Trigger sind Bestandteile des folgenden Abschnitts. Der Index ermöglicht es, Suchabfragen mit Ercatons zu realisieren. Abschließend wird auf ausgewählte Konzepte zur Anzeige, wie den Katalog und Resource-Ercatons, mit deren Hilfe es möglich ist binäre Dateien einzubinden, eingegangen. Außerdem wird erklärt, wie das Erscheinungsbild von Ercatons verändert werden kann.

6.1 Ausgewählte Konzepte der Ercatons-Programmierung

Eines der wichtigsten Konzepte der Ercatons-Programmierung ist die Vererbung. Im Folgenden wird anhand von Formeln und Beispielen gezeigt, wie sie mit Ercatons implementiert werden kann. Es wird sowohl ein objektorientiertes, als auch ein Thing-orientiertes Beispiel vorgestellt. Mehrwertige Attribute können in Ercatons mit Hilfe von sogenannten *fields* erzeugt werden. Dies wird im folgenden Abschnitt, ebenso wie deren grafische Anzeige, erklärt. Abschließend wird erläutert wie Aktionen in Ercatons mittels Java implementiert werden können.

6.1.1 Vererbung zwischen Ercatons

Die objektorientierte Vererbung entspricht einem Sonderfall des *XOperators*. Er ist ein von der Living Pages Research GmbH entwickeltes Konzept, das eine einfache Algebra zwischen XML-Dokumenten zur Verfügung stellt. Hiermit ist es möglich XML-Dokumente zu vergleichen und zusammen zu führen [Liv09b]. Die Vererbung entspricht somit einer Addition. Seien zum Beispiel a und b Ercatons (alle Ercatons außer Resource-Ercatons) so gelten die folgenden Gleichungen:

$$a \equiv a + b \Leftrightarrow a \; erbt \; von \; b \tag{6.1}$$

$$a \equiv a + a \tag{6.2}$$

$$a - a \equiv 0 \tag{6.3}$$

$$(a - b) + b \equiv a \tag{6.4}$$

$$\exists a,b : a + b \neq b + a \tag{6.5}$$

Um eine Vererbung zu implementieren ist es nötig, die Eigenschaften des Ercatons, welches erbt (*Clone*) mit denjenigen Eigenschaften des Eratons von dem geerbert wird (*Clonebase*), synchron zu halten. Das heißt, dass Änderungen an der Clonebase auch Änderungen am Clone bewirken. Auf diese Weise ist es, wie auch bei der Prototyp-basierten Programmierung, möglich, viele Ercatons gleichzeitig durch Änderung der Clonebase zu editieren. Eine Clonebase entspricht hierbei dem Prototyp der Prototyp-basierten Programmierung. Gleichung 6.1 verdeutlicht dieses Konzept. Eine Ercato Engine darf keinen Unterschied zwischen einem Ercaton a und einem Ercaton a' machen, wenn die linke Seite der Gleichung 6.1 für a' und b erfüllt ist und a ein Clone von b ist. Das Ercaton b, von dem das Ercaton a kopiert wurde, ist Clonebase von a. Sollten Änderungen an der Clonebase vorgenommen werden, gibt die Ercato Engine diese Änderungen an den Clone weiter.
Dieses Modell unterstützt nicht nur Fälle, in denen keine Instanz einer anderen gleicht, sondern auch klassische objektorientierte Techniken, wie zum Beispiel Objekte, die von ihren Vorgängern erben oder Objekte, die von Klassen instantiiert werden. Objekte, die gemeinsame Teile besitzen zählen ebenfalls zu den unterstützten Techniken [ILvW04].
Die Anzahl der Clonebases pro Ercaton ist auf eine einzige beschränkt. Jedoch kann ein Ercaton mehrere sogenannte *Bases* haben. Eine Base entspricht einer Clonebase, jedoch

mit dem Unterschied, dass nach Erzeugung des Ercatons, das von der Base geerbt hat, die Verbindung zwischen Base und Clone abbricht. Das bedeutet, dass eine Änderung der Base keine Änderung des Clones bewirkt. Falls der einzige Zweck eines Ercatons darin besteht, als Clonebase zu dienen, so kann der Typ *prototype* verwendet werden. Dies ist vergleichbar mit einer abstrakten Klasse in der OrP.
Gleichung 6.2 zeigt, dass ein Ercaton idempotent ist. Das heißt, eine Verknüpfung eines Ercatons a mit sich selbst ergibt wieder Ercaton a. Erbt also Ercaton a von sich selbst, so verändert es sich nicht.
Eine Subtraktion (Gleichung 6.3) zeigt die Unterschiede zwischen zwei Ercatons. In diesem Fall existiert natürlich zwischen zwei identischen Ercatons keine Differenz.
Wenn Ercaton b von Ercaton a subtrahiert wird und anschließend von Ercaton b erbt, ergibt dies Ercaton a (Gleichung 6.4).
Gleichung 6.5 besagt, dass keine Kommutativität gilt. Erbt zum Beispiel Ercaton a von Ercaton b, so ist das resultierende Ercaton nicht mit demjenigen identisch, das entsteht wenn Ercaton b von Ercaton a erbt.

Objektorientiertes Beispiel

Ein objektorientiertes Beispiel der Vererbung zeigt das folgende Codebeispiel. Hier werden Eigenschaften oder auch Funktionen überschrieben. Codebeispiel 6.1 zeigt ein stark vereinfachtes Ercaton für ein Caipirinha-Rezept. Es besitzt eine Ercaton-ID und vier verschiedene Zutaten: „Rohrzucker“, „Limetten“, „zerstoßenes Eis“ und „Rum“. In Codebeispiel 6.2 ist ein Ercaton für ein Caipiroska-Rezept dargestellt, welches das Caipirinha-Rezept als Clonebase besitzt und alle Eigenschaften, das heißt Zutaten, von ihm erbt. In diesem Fall sind beide Ercatons, bis auf die Ercaton-ID, welche immer eindeutig sein muss, identisch. Wird das Caipirinha-Ercaton verändert, so ändert sich auch das Caipiroska-Ercaton. Dies ist hier von Vorteil wenn die Zutaten angepasst werden, da Caipirinha und Caipiroska bis auf die Art des Alkohols identisch sind.

```
1 <recipe>
2   <erc:id>~sample/caipirinha</erc:id>
3   <ingredients>
4     <ingredient1> Rohrzucker </ingredient1>
5     <ingredient2> Limetten </ingredient2>
```

```
6       <ingredient3> zerstoßenes Eis </ingredient3>
7       <ingredient4> Rum </ingredient4>
8     </ingredients>
9   </recipe>
```

Quellcode 6.1: Das Caipirinha-Ercaton

```
1   <recipe>
2       <erc:clone>~sample/caipirinha</erc:clone>
3       <erc:id>~sample/caipiroska</erc:id>
4   </recipe>
```

Quellcode 6.2: Das Caipiroska-Ercaton als Clone des Caipirinha-Ercatons

Da ein Caipiroska im Gegensatz zu Caipirinha nicht „Rum“, sondern „Wodka“ beinhaltet, wird das Beispiel verbessert und in Codebeispiel 6.3 ein nicht-leerer Clone mit dem zusätzlichen Tag `<ingredient4>`, welcher „Wodka“ als Wert beinhaltet, erstellt. Somit wird `<ingredient4>` nicht mehr von Caipirinha geerbt, sondern durch den, in Caipiroska definierten, Wert überschrieben. Die restlichen Zutaten bleiben hiervon unberührt. Wird `<ingredient1>` bis `<ingredient3>` in Caipirinha verändert, so ändern sich auch die entsprechenden Werte in Caipiroska.

```
1   <recipe>
2       <erc:clone>~sample/caipirinha</erc:clone>
3       <erc:id>~sample/caipiroska</erc:id>
4       <ingredient4> Wodka </ingredient4>
5   </recipe>
```

Quellcode 6.3: Das Caipiroska-Ercaton als nicht-leerer Clone des Caipirinha-Ercatons

Thing-orientiertes Beispiel

Ein Thing-orientiertes Beispiel wird in Abbildung 6.1 gezeigt. Es sind die beiden Ercatons `~bsp/hase` und `~bsp/schleife` dargestellt. Jedes dieser Ercatons repräsentiert ein Thing, das heißt eine Entität der realen Welt: einen Hasen und eine Schleife. Um einen Hasen mit Schleife zu erstellen, die beiden Ercatons also zu verbinden, wird ein neues Ercaton (`~bsp/hase-mit-schleife`) erzeugt, welches identisch mit dem Ercaton `~bsp/schleife` ist, zusätzlich aber einen `<erc:clone>`-Tag besitzt, welcher

die Ercaton-ID von `~bsp/hase` beinhaltet. Somit erbt das neu erstellte Ercaton alle Eigenschaften des Hasen-Ercatons.

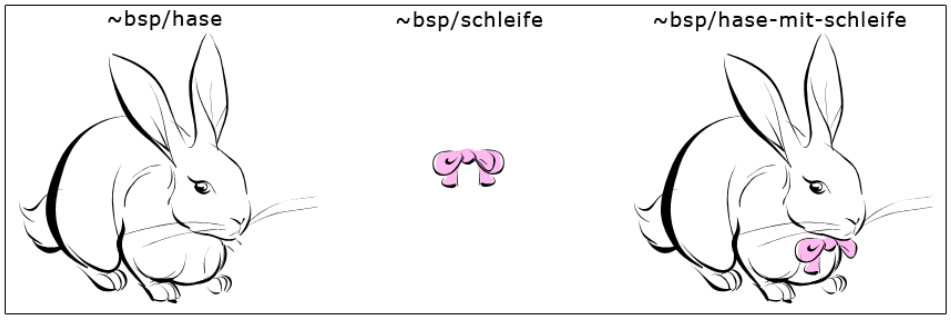


Bild 6.1: Ein „Schleifen“-Ercaton erbt von einem „Hasen“-Ercaton

6.1.2 Mehrwertige Attribute

Mehrwertige Attribute sind Attribute mit einer variablen Anzahl von Werten. Sie werden zum Beispiel benötigt um Cocktail-Rezepte zu erstellen, da die Menge an Zutaten von Rezept zu Rezept verschieden sein kann. In Ercatons werden mehrwertige Attribute mit Hilfe von sogenannten *fields* modelliert. Sie erlauben eine variable Anzahl von Zeilen. Das Codefragment 6.4 erzeugt in einem Ercaton ein mehrwertiges Attribut. Die wichtigen Elemente sind hierbei `erc:inherited-as="vector"` in Zeile 4, welches das Listenelement als Vektor kennzeichnet, der in diesem Fall aus den beiden Einträgen *amount* und *name* besteht. Die Länge dieses Vektors ist nicht beschränkt. Außerdem verweisen `erx:list-ref="ingredients"` und `erx:nodename="ingredient"` (Zeile 3) unter `erx:fields` in Zeile 14 auf die entsprechende Liste.
Das Element `erx:list` deklariert einen Listen-Prototyp, welcher dazu verwendet wird editierbare Felder einer Liste zu definieren und neue Einträge in einer leeren Liste zu erzeugen. Das Attribut `nodename` legt den Namen des Listen-Member Knotens fest (vergleiche hierzu das Manpage-Ercaton `/man/std-xsl` zu `/bin/std-xsl` bzw. `/builtin/default.xsl`). Er dient dazu, das letzte Element der Liste zu finden um neue Zeilen dort anzuhängen. Außerdem muss beim Hinzufügen von Werten die XML-Baumstruktur erweitert werden, wobei `nodename` festlegt welches Element hierfür als Prototyp dient und repliziert wird.

```
<cocktail>
    <name erx:field-name="Cocktail Name">Caipirinha</name>
    <ingredients erx:list-ref="ingredients" erx:nodename="ingredient">
      <ingredient erc:inherited-as="vector">
          <amount erx:field-ref="colamount"/>
          <name erx:field-ref="colname"/>
      </ingredient>
  </ingredients>
</cocktail>

<erx:fields>
    <erx:field erc:name="colname" field-name="Zutat" tr="string"/>
    <erx:field erc:name="colamount" field-name="Menge" tr="string"/>
    <erx:list erc:name="ingredients" field-name="Zutaten" edit="true"
       nodename="ingredient">
      <ingredient>
          <amount erx:field-ref="colamount"/>
          <name erx:field-ref="colname"/>
      </ingredient>
  </erx:list>
</erx:fields>
```

Quellcode 6.4: Mehrwertige Attribute mit „fields“

Grafische Anzeige mehrwertiger Attribute

Ein mehrwertiges Attribut wird in Ercatons mit Hilfe des sogenannten *Table-Control* angezeigt. Das Table-Control ist Bestandteil der Erxlib und wird im Ercaton `/builtin/default.xsl` definiert. Außerdem wird im Ercaton `/system/applets.xsl` die Java-Klasse *com.livis.forms.applet.TableApplet.class* aufgerufen, der Parameter, wie zum Beispiel die Hintergrundfarbe der Liste, übergeben werden. Das Codefragment 6.4 erzeugt eine grafischen Anzeige wie in Abbildung 6.2 zu sehen ist.

Bild 6.2: Grafische Anzeige eines mehrwertigen Attributes mit Table-Control

6.1.3 Builtin-Ercaton adder

Das Builtin-Ercaton `/builtin/adder` stellt einen reservierten Platzhalter im Namensraum aller Ercatons dar und sammelt Informationen, die per `erc:add` Attribut indiziert wurden. Es wird zum Beispiel zur Implementierung von Bäumen mit Referenzzählern verwendet. Das Codebeispiel 6.5 soll die Notwendigkeit dieses Builtin-Ercatons verdeutlichen. Es zeigt drei verschiedene Ercatons: `~foo/bar1`, `~foo/bar2` und `~foo/bar3`. Hierbei sollen die Einkaufswägen 1 und 2 im Ercaton `~foo/prices` indiziert und deren Summen aufgelistet werden. Der genaue Mechanismus zur Indizierung wird in Abschnitt 6.3.2 erläutert. In diesem Beispiel ist es wichtig zu bemerken, dass „Einkaufswagen 1“ doppelt auftritt. In `~foo/prices` ist deshalb eine eindeutige ID-Zuordnung nicht mehr möglich, weshalb die Engine die Summen dem reservierten Ercaton `/builtin/adder` zuschlägt. Codebeispiel 6.6 zeigt, wie Summen im `~foo/prices` Ercaton gespeichert und den Einkaufswägen zugeordnet werden. Die Ercaton-ID zu den entsprechenden Summen lautet `/builtin/adder` und „Einkaufswagen 1“ tritt jetzt nur einmal auf, jedoch mit der Summe über beide Einkaufswägen.

```
1 <erc:id>~foo/bar1</erc:id>
2 <process erc:index="~foo/prices">Einkaufswagen 1</process>
3 <price erc:add="~foo/prices#total">17.59</price>
4
5 <erc:id>~foo/bar2</erc:id>
6 <process erc:index="~foo/prices">Einkaufswagen 1</process>
7 <price erc:add="~foo/prices#total">2.49</price>
8
```

```
9  <erc:id>~foo/bar3</erc:id>
10 <process erc:index="~foo/prices">Einkaufswagen 2</process>
11 <price erc:add="~foo/prices#total">9.99</price>
```

Quellcode 6.5: Ercatons mit „erc:add“ Attribut

```
1 id: /builtin/adder; process: Einkaufswagen 1; total: 20.08
2 id: /builtin/adder; process: Einkaufswagen 2; total:  9.99
```

Quellcode 6.6: Speicherung der Summen

6.1.4 Aktionen

Aktionen können in Ercatons mit Hilfe einer Programmiersprache, wie zum Beispiel Java, implementiert werden. Struktur und Verhalten des Ercatons sind auf diese Weise getrennt. Die Struktur wird im Ercaton selbst per XML definiert, während das Verhalten extern in einer Java-Klasse implementiert ist. Der Vorteil besteht darin, dass die Java-Klasse auch von anderen Ercatons, welche eine entsprechende Signatur besitzen, angesprochen werden kann. Im Folgenden wird ein Beispiel für einen Zähler mit einer in Java implementierten Aktion beschrieben.

6.1.4.1 Zähler-Ercaton

In Codebeispiel 6.7 ist ein Zähler-Ercaton mit einer Aktion zu sehen, die in Java implementiert ist. Das `<count>`-Element enthält einen numerischen Wert, der den aktuellen Wert des Zählers festlegt. Das Tag `<erc:object>` in Zeile 4 bindet ein Java-Objekt an das Ercaton. Mit Hilfe von `<erc:archive>` wird das Code-Archiv mit der entsprechenden Java-Klasse eingebunden, welche über `<erc:class>` definiert wird. Es ist die Deklarationsschicht und die Implementierungsschicht einer Aktion zu unterscheiden:

Deklarationsschicht

Das Tag `<erc:action>` im Codebeispiel 6.7, Zeile 8, deklariert eine Aktion *main* im aktuellen Ercaton, die von außerhalb zum Beispiel per `~sample/count!main(amount=1)` aufgerufen werden kann. Mit `<erc:arg>` wird ein Argument für diese Aktion definiert, welche den Standardwert „1“ besitzt. Da die Methode *main* immer als Standard aufgerufen

wird und ein Defaultwert definiert ist, genügt der Aufruf von `~sample/count` um den Counter um „1“ zu erhöhen.

Implementierungsschicht

Die Implementierungsschicht beginnt in Zeile 10 des Codebeispiels 6.7. Hier spezifiziert `<erc:native>` die Sprache Java als Implementierungssprache und `<erc:method>`, sowie `<erc:parameter>` mappen die Aktion auf eine Java-Implementierung. Die Methode wird hierbei durch `<erc:method>` und die Parameter dieser Methode durch `<erc:parameter>` definiert. Als Parameter wird der Methode das Argument der Aktion weitergereicht, also in diesem Fall „amount“. Über den Klassennamen des `<erc:object>`-Elements (Zeile 4), die Methode des `<erc:method>`-Elements und alle `<erc:parameter>` wird eine exakte Java-Methodensignatur berechnet.

```
1  <counter>
2     <erc:id>~sample/count</erc:id>
3     <count>0</count>
4     <erc:object lang="Java">
5          <erc:archive>~sample/lib.jar</erc:archive>
6          <erc:class>sample.Counter</erc:class>
7     </erc:object>
8     <erc:action name="main">
9          <erc:arg name="amount">1</erc:arg>
10         <erc:native lang="Java">
11                 <erc:method>increment</erc:method>
12                 <erc:parameter name="amount" type="int"/>
13         </erc:native>
14    </erc:action>
15 </counter>
```

Quellcode 6.7: Ein Zähler-Ercaton mit Java Implementierung

6.1.4.2 Java-Klasse

Ercatons werden aus Java als Objekt angesprochen (*com.ercato.core.ErcatonObject*). Jedes Ercaton, welches in der Java-Umgebung erscheint, das heißt jede Instanz von *com.ercato.core.ErcatonObject*, stellt nur eine Kopie des eigentlichen Ercatons dar. Diese

Kopie wird verändert und abschließend von der Engine über das eigentliche Ercaton geschrieben. Codebeispiel 6.8 zeigt die Java-Klasse *Counter* für das Zähler-Ercaton in Beispiel 6.7. Das Beispiel beinhaltet drei Methoden, welche im Folgenden genauer erklärt werden:

evaluateElement(EvaluationContext ec, String tag, String ns)

Die Methode *evaluateElement* überschreibt die Methode *evaluateElement* der Klasse *ErcatonObject*. Sie wird beim Traversieren des XML-Baums für jedes Element aufgerufen. Durch Überschreiben der Methode ist es möglich nach Teilen zu suchen, die interessant sind. Der *EvaluationContext* enthält das aktuell traversierte Element und die Schachtelungstiefe im Baum. Die beiden String-Argumente geben den Namen der zu suchenden Elemente (*tag*), sowie den Namensraum (*ns*) an. In Beispiel 6.8, Zeile 14, wird im XML-Baum, das heißt in dem Ercaton von dem aus die Klasse aufgerufen wurde, das „count“-Element gesucht. Anschließend wird mit `getTextNode(false)` der aktuelle Knoten in die private Variable „counter“ eingelesen. Der Parameter „false“ gibt hierbei an, dass „null“ zurückgegeben werden soll, falls kein Knoten existiert. In der Variable „counter“ steht nun ein Text, welcher in Codezeile 16 in einen Integer Wert umgewandelt und in der privaten Variable „count“ gespeichert wird.

increment(int amount)

Die öffentliche Methode *increment* in Beispiel 6.8, Zeile 9, die vom Counter-Ercaton direkt aufgerufen wird, erhöht die private Variable „count“ um den Wert, der ihr mit Hilfe des Parameters „amount“ übergeben wurde und ruft anschließend die Methode *approveErcaton* auf.

approveErcaton()

In *approveErcaton* (Codebeispiel 6.8 ab Codezeile 18) wird der Wert der „count“ Variable in einen Textknoten umgewandelt und in „counter“ gespeichert. Da die Engine Änderungen auf XML-Ebene per *org.w3c.dom-API* noch nicht automatisch registriert, muss die Methode *ErcatonObject.touch()* aufgerufen werden. Sie teilt der Engine mit, dass die entsprechende Kopie des Ercatons modifiziert wurde und bei Gelegenheit das eigentliche Ercaton aktualisiert werden sollte. Wie die Engine dies im Hintergrund

verarbeitet wird an dieser Stelle nicht näher erläutert.

```
1  package sample;
2  import com.ercato.core.*;
3  import org.w3c.dom.Text;
4
5  public class Counter extends ErcatonObject implements Action {
6     private Text counter;
7     private int count;
8
9     public void increment (int amount) {
10         count += amount;
11         if (amount != 0) approveErcaton ();
12    }
13    protected void evaluateElement (EvaluationContext ec, String tag,
          String ns) {
14         if (!"count".equals (tag)) return;
15         counter = ec.getTextNode (false);
16         count = Integer.parseInt (counter.getDate ());
17    }
18    protected void approveErcaton () {
19         counter.setDate (String.valueOf (count));
20         touch ();
21    }
22 }
```

Quellcode 6.8: Java-Klasse „Counter“

6.2 Konzept zur Zugriffskontrolle

Das Ercatons-Zugriffsmodell realisiert sowohl eine Rollen-basierte, als auch die, aus der OrP bekannte, Paket- oder Modul-basierte Zugriffskontrolle (zum Beispiel *private*, *package-private*, *public*, usw.). Letztere wird durch die Verwendung von Paketnamen als Rollen umgesetzt.
Durch folgende, rollenbasierte Zugriffsrechte werden Status und Aktionen von Ercatons geschützt: Es ist möglich für jedes Ercaton anzugeben wer es lesen (**r**ead), schreiben (**w**rite), navigieren (**b**rowse) oder ausführen (e**x**ecute) darf. Außerdem existiert das

sogenannte *s-bit*. Mit Hilfe dieses Bits ist es möglich, eine Aktion mit den Rechten des Besitzers dieser Aktion auszuführen (vergleiche s-bit unter UNIX). Somit kann eine Aktion ausgeführt werden, für die der Benutzer zunächst keine Rechte hat. Sie werden ihm durch das s-bit kurzfristig zugeteilt und nach Beendigung wieder entzogen. Ein zweites s-bit (t) legt fest, dass die entsprechende Aktion ausführbar ist und das aktuelle Ercaton verändern darf. Mit einem Browse-Recht wird automatisch auch das Leserecht vergeben, nicht jedoch umgekehrt.

Role-Ercaton

In einem *Role-Ercaton* wird definiert, wer wem eine Rolle zuweist oder zuweisen kann. Es gibt hierbei keine einschränkenden Rollen und es ist nicht möglich sich gegen die Zuweisung einer Rolle zu wehren. Es ist sinnvoll zwei Arten von Role-Ercatons zu unterscheiden: Zum einen die *funktionalen Rollen*, die als Zugriffsrechte in den Ercatons verwendet werden und zum anderen *Gruppen*, welche mehrere Personen rechtemäßig zusammenfassen. Technisch sind beide Arten jedoch identisch. Das Codebeispiel 6.9 zeigt ein Role-Ercaton für die Rolle *Supervisor*. Mit `<erc:granted-for role="~max"/>` wird dem Benutzer `~max` diese Rolle zugewiesen. Anstelle eines Benutzers können hier auch Rollen angegeben werden. Außerdem ist es möglich mehrere `<erc:granted-for>`-Tags hintereinander zu stellen und somit mehreren Benutzern die entsprechende Rolle zuzuweisen.
Mit Hilfe des `<erc:permission>`-Tags ist es möglich einer Rolle (hier `~admin`) das Recht zu geben, Änderungen an der Rolle durchzuführen, und damit zu steuern wer die Rolle vergeben darf. Wird dies nicht definiert, so darf nur die Rolle `~root` Rollen hinzufügen oder löschen.

```
<role>
 <erc:id>~test/roles/supervisor</erc:id>
 <erc:type>role</erc:type>
 <erc:permission role="~admin">rwb</erc:permission>
 <name>Supervisor</name>
 <erc:role>
   <erc:granted-for role="~max"/>
 </erc:role>
</role>
```

Quellcode 6.9: Ein Role-Ercaton

Die beiden Role-Ercatons `~admin` und `~root` sind Teil der Builtin-Ercatons (vgl. Abschnitt 5.2.3) und werden bei der Installation der Engine mit installiert. Sie sind beide der Engine bekannt. Die Rolle `~root` hat hierbei Superuserrechte, das heißt, Mitlieder dieser Rolle dürfen alles, wohingegen `~admin` keine Sonderrechte besitzt, ohne dass sie ihr ausdrücklich, durch oben genanntes `<erc:permission>`-Tag, zugewiesen worden sind. Desweiteren existieren die beiden Rollen `~any` und `~anonymous`. Erstere gilt für alle Benutzer, die an der Engine angemeldet sind, ohne Rücksicht auf ihren Namen. Sie besitzen automatisch die Rolle `~any`. Nicht angemeldete Benutzer hingegen erhalten die Rolle `~anonymous`. Wird in einem Ercaton das Zugriffsrecht `<erc:permission role="~anonymous">rxb</erc:permission>` definiert, so hat jeder, egal ob angemeldet oder nicht, Zugriff auf dieses Ercaton. Er kann es lesen (r), ausführen (x) und es wird ihm in Suchabfragen angezeigt (b).

User-Ercaton

Ein User-Ercaton ist ein Untertyp eines Role-Ercatons und kann einen Menschen (Benutzer, Besitzer), eine Gruppe oder eine geschützte Domäne, wie zum Beispiel ein Software-Paket, darstellen. Das Codebeispiel 6.10 zeigt ein User-Ercaton für den Benutzer Max Mustermann. Durch das Tag `<erc:permission role="~admin">rwb</erc:permission>` wird der Rolle `~admin` das Recht zugewiesen, dieses Ercaton zu lesen und zu editieren. Hierdurch können Benutzer, die die Rolle `~admin` besitzen, zum Beispiel die Adressdaten von Max Mustermann ändern.

```
1 <user>
2   <erc:id>~max</erc:id>
3   <erc:type>user</erc:type>
4   <erc:permission role="~admin">rwb</erc:permission>
5   <person>
6      <first-name>Max</first-name>
7      <last-name>Mustermann</last-name>
8   </person>
9 </user>
```

Quellcode 6.10: Ein User-Ercaton für den Benutzer Max Mustermann

6.3 Ausgewählte Konzepte zur Persistenz und Abfrage

Im folgenden Abschnitt wird zunächst der Persistenzmechanismus der Ercatons vorgestellt und anschließend der sogenannte *Index* erläutert. Ercatons beruhen auf dem Prinzip, dass Persistenz und strukturierte Abfragen orthogonal sind und deshalb, wie es zum Beispiel auch im WWW der Fall ist, getrennt voneinander behandelt werden sollen. Wie dies mit Index-Attributen, Index-Ercatons und Query-Ercatons realisiert ist, wird hier erklärt. Trigger fügen einem Ercaton autonomes Verhalten hinzu und werden ebenfalls in diesem Abschnitt behandelt.

6.3.1 Skizze des internen Persistenzmechanismus

Die Persistenz der Ercatons wird über einen eigenen Transaktions-Layer realisiert, welcher direkt auf dem Filesystem aufsetzt und sich unterhalb des Verzeichnisses *erc-fs-db* befindet. Persistenzinformationen liegen hier zum Beispiel in einer redundanzfreien XML-Darstellung. Das Persistenzlayer ist so konzipiert, dass unter anderem *Echtzeitsnapshots* im laufenden Betrieb möglich sind. Dies wird durch Hardlinks vom aktuellen in ein weiteres Verzeichnis realisiert.
Die Ordner- und Dateibezeichnungen in der *erc-fs-db* setzen sich wie folgt zusammen: Ein Ercaton mit der ID `.../id` wird sich als Datei mit der Bezeichnung `.../z#<nn>/id,<v>,<tid>` wiederfinden. Hierbei ist `<nn>` ein Hashwert der ID, das heißt er ist für das aktuelle Ercaton konstant, aber ansonsten zufällig. `<v>` ist die Version des Ercatons, die für unversionierte Ercatons leer bleibt und `<tid>` stellt die Transaktions-ID dar, an der ein Ercaton gerade teilnimmt. Das Tag ist außerhalb von Transaktionen leer. Transaktionen laufen isoliert voneinander ab und können Ercatons, die sich in anderen Transaktionen befinden, nicht sehen. Auf weitere technische Details des Persistenzmechanismus wird in dieser Studienarbeit nicht eingegangen.

6.3.2 Index

In der OOP ist das Abbilden von Objekten auf relationale Datenbanken ein ernstzunehmendes Problem. Objekte werden über ihre Verbindungen durchlaufen, wohingegen im relationalen Paradigma Datenzeilen in Tabellen verbunden werden. Diese fundamentalen

Unterschiede führen zu einer nicht-idealen Kombination von Objekt- und relationalen Technologien, dem sogenannten „Object-relational impedance mismatch“ [Amb03]. Auch organisches Programmieren macht hier keine Ausnahme [ILvW04]. Folgende Regel wird in [ILvW04] genannt: *„Persistenz und strukturierte Abfragen sind orthogonal und sollten unabhängig voneinander implementiert werden.“* Als Muster-Beispiel wird das Internet genannt, das aus einem Netzwerk aus Webseiten für die Persistenz und aus Suchmaschinen, wie zum Beispiel „Google“ für Abfragen besteht.
Das Ercato Programmiermodell bietet hierzu einen Mechanismus, der aus den folgenden drei Bausteinen besteht: *Index-Ercatons*, *Index-Attribute* in beliebigen Ercatons, sowie *Query-Ercatons* für strukturierte Abfragen über Relationen.

Codebeispiel 6.11 zeigt das Caipirinha-Ercaton aus Abschnitt 6.1.1 mit Name und Index-Attributen. Per *erc:index* wird der Textinhalt des jeweiligen Elements indiziert. Es ist ein universelles XML-Attribut und an jedem Element erlaubt. Der Name des Index wird automatisch aus dem Tag bestimmt, es sei denn mit `#index-name` wird ein alternativer Index-Name spezifiziert, wie zum Beispiel in Zeile 8 mit `erc:index="~sample/recipes#alcohol"`. Der Wert des Tags `<ingredient4>` wird nicht im Index „ingredient4“, sondern im Index „alcohol“ gespeichert. Codebeispiel 6.12 zeigt ein hierzu passendes Index-Ercaton für Cocktail-Rezepte. Es beinhaltet einen Index für den Cocktail-Namen in Zeile 5, sowie vier weitere Indizes für die Zutaten.

```
1  <recipe>
2    <erc:id>~sample/caipirinha</erc:id>
3    <name erc:index="~sample/recipes">Caipirinha</name>
4    <ingredients>
5      <ingredient1 erc:index="~sample/recipes">Rohrzucker</ingredient1>
6      <ingredient2 erc:index="~sample/recipes">Limetten</ingredient2>
7      <ingredient3 erc:index="~sample/recipes">zerstoßenes Eis</
           ingredient3>
8      <ingredient4 erc:index="~sample/recipes#alcohol">Rum</ingredient4>
9    </ingredients>
10 </recipe>
```

Quellcode 6.11: Das Caipirinha-Ercaton mit Index-Attributen

Um jetzt zum Beispiel alle Cocktails abzufragen, die die Zutat '"Rum“ als Alkohol beinhalten, kann das Service-Ercaton `/bin/simplequery` verwendet werden. Dies wird

```
<recipes>
  <erc:type>index</erc:type>
  <erc:id>~sample/idx/recipes</erc:id>
  <erc:index>
    <erc:name>name</erc:name>
  </erc:index>
  <erc:index>
    <erc:name>ingredient1</erc:name>
  </erc:index>
  <erc:index>
    <erc:name>ingredient2</erc:name>
  </erc:index>
  <erc:index>
    <erc:name>ingredient3</erc:name>
  </erc:index>
  <erc:index>
    <erc:name>alcohol</erc:name>
  </erc:index>
</recipes>
```

Quellcode 6.12: Ein Index-Ercaton für Cocktail-Rezepte

in Codebeispiel 6.13 dargestellt. Das Ergebnis dieser Abfrage ist ein Listen-Ercaton. Es beinhaltet eine Liste aller IDs derjenigen Ercatons, welche das entsprechende Constraint erfüllen. Außerdem enthält es zusätzliche indizierte Informationen (in diesem Fall also die restlichen Zutaten, sowie den Cocktail-Namen). Es werden jedoch keine Informationen gefunden für die der Benutzer nicht die entsprechenden Rechte besitzt. Das Recht *navigierbar (b)* ist mindestens nötig.

```
/bin/simplequery(index="~sample/idx/recipes" name="alcohol" value="Rum
    ")
```

Quellcode 6.13: Abfrage aller Cocktails mit Rum als alkoholische Zutat

Mit Hilfe eines Query-Ercatons, wie in Codebeispiel 6.14 gezeigt, kann diese Abfrage ebenfalls realisiert werden. In Zeile 5 wird mit `<erc:id-ref>` das Index-Ercaton, in dem gesucht werden soll, eingebunden. Innerhalb des `<columns>`-Tags werden die anzuzeigenden Spalten mit Breite (Zeile 11) und Sortierung (Zeile 10) spezifiziert. Mit Hilfe des `<filters>`-Tags ab Zeile 15 ist es möglich, Filterkriterien zu definieren. In diesem Beispiel werden nur Ercatons angezeigt, welche den Wert „Rum“ im Tag `<alcohol>` beinhalten. Query-Ercatons sind sehr mächtig und besitzen zahl-

reiche Funktionaliäten, die in dieser Studienarbeit aber nicht genauer beschrieben werden.

```
<query>
  <erc:id>~sample/query/alcohol</erc:id>
  <query>
    <index>
      <erc:id-ref>~sample/idx/recipes</erc:id-ref>
    </index>
    <columns>
      <column>
        <name erx:display="Cocktail-Name">name</name>
        <sort>asc</sort>
        <width/>
        <idrefcol>id</idrefcol>
      </column>
    </columns>
    <filters>
      <or>
        <and>
          <column>alcohol</column>
          <value>Rum</value>
        </and>
      </or>
    </filters>
  </query>
</query>
```

Quellcode 6.14: Ein Query-Ercaton für Cocktail-Rezepte

6.3.3 Trigger

Trigger (engl.: Auslöser) fügen Ercatons autonomes Verhalten hinzu. Ercatons können somit auf Ereignisse wie Änderungen oder Ablauf einer bestimmten Zeitspanne reagieren. Mit folgender Codezeile wird beispielsweise ein Trigger definiert, welcher auf eine Änderung mit dem Aufruf des `/bin/check` Ercatons reagiert:

```
<erc:trigger event="on-change"> /bin/check </erc:trigger>
```

Hiermit können dann zum Beispiel eingegebene Formulardaten überprüft werden.
Falls bei der Abarbeitung Fehler auftreten, deaktiviert die Ercato Engine Trigger, die sich wiederholen (z.B. „jede Minute“). Dadurch wird ein stabiler Engine Betrieb gewährleistet. Codebeispiel 6.15 zeigt einen `<trigger>`-Tag, welcher einen Trigger mit Zeitintervall definiert. Es wird alle drei Minuten die Aktion *test* aufgerufen.

```
<erc:trigger event="time-interval" name="test">
  <erc:time>3</erc:time>
  <erc:action-ref>!test</erc:action-ref>
</erc:trigger>
```

Quellcode 6.15: Ein Trigger mit Zeitintervall

6.4 Ausgewählte Konzepte zur Anzeige

Zu den ausgewählten Konzepten zur Anzeige zählt unter anderem der Katalog. Dieser dient dazu Ercatons in einer Weboberfläche einzugliedern und darauf aufbauend eine Navigation zu realisieren. Der folgende Abschnitt zeigt, wie Ercatons in den Katalog eingebunden werden, eine Navigation erstellt und diese anschließend in die Weboberfläche integriert werden kann.
Anschließend wird erklärt wie das Erscheinungsbild von Ercatons verändert werden kann und als Abschluss dieses Abschnitts folgt eine Beschreibung der sogenannten *Resource-* und *Referenz-Ercatons*, mit deren Hilfe es möglich ist binäre Dateien, wie zum Beispiel PDF-Dateien oder Bilder, in Ercatons einzubinden.

6.4.1 Katalog

Der Katalog ist ein Ercatons-spezifisches Konzept und wird über das `<erc:catalog>`-Tag, sowie ein passendes Index-Ercaton gesteuert. Er ist ein hilfreiches Mittel um Ercatons in einer Weboberfläche einzugliedern und in Kategorien zu unterteilen. Mit Hilfe des Service-Ercatons `/bin/catalog` lässt sich darauf aufbauend zum Beispiel schnell und einfach eine Navigation realisieren. In Abbildung 6.3 ist eine Beispiel-Navigation dargestellt.

Bild 6.3: Eine Beispiel-Navigation

6.4.1.1 Index-Ercaton für den Katalog

Um einen Katalog zu realisieren muss zunächst ein Index-Ercaton erstellt werden. Es muss mindestens einen Index für *category* und *name* beinhalten. Weitere Indizes sind erlaubt. Außerdem wird ein Index für Zahlen benötigt, der mit dem `<erc:adder>`-Attribut (Zeile 12) gekennzeichnet ist. Dieser dient dazu, den Zähler für die Kategorien zu erhöhen oder zu reduzieren. Beinhaltet zum Beispiel eine Kategorie zwei Ercatons, so wird sie nur einmal angezeigt und der Zähler beinhaltet den Wert „2“. Werden die Ercatons wieder gelöscht, wird auch der Zähler um die entsprechende Anzahl erniedrigt. Sobald er den Wert „0“ erreicht, wird die Kategorie wieder aus dem Katalog entfernt, da sie keine Ercatons mehr beinhaltet. In Codebeispiel 6.16 ist das Index-Ercaton `~recipes/catalog` mit den Attributen *name* und *category* dargestellt. Der Index *RefCount* stellt den Index für den Zähler dar.

```
1  <erc:type>index</erc:type>
2  <erc:id>~recipes/catalog</erc:id>
3    <erc:index>
4    <erc:name erx:display="Kategorie">category</erc:name>
5  </erc:index>
6  <erc:index>
7    <erc:name erx:display="Name">name</erc:name>
```

```
8   </erc:index>
9   <erc:index>
10    <erc:name>RefCount</erc:name>
11    <erc:index-type>int</erc:index-type>
12    <erc:adder/>
13  </erc:index>
```

Quellcode 6.16: Das Index-Ercaton recipes/catalog

6.4.1.2 Einbinden von Ercatons in den Katalog

Mit der Definition eines `<erc:catalog>`-Tags innerhalb eines beliebigen Ercatons ist es möglich dieses in den Katalog einzubinden. Ein Beispiel hierzu wird in Codebeispiel 6.17 gezeigt. Das Tag `<erc:catalog>` stellt einen abstrakten Mechanismus dar, der unter anderem einen Indexeintrag erzeugt. Er benötigt die ID eines Index-Ercatons, die ihm per *id-ref* mitgeteilt wird. Das Attribut *id-ref* hat jedoch nur im `<erc:catalog>`-Element diese Bedeutung. Mit dem XML-Attribut *erc:index* wird der Textinhalt des Elements indiziert (vgl. 6.3.2).

```
1 <erc:catalog category="/Rezepte/Nudeln/" id-ref="~recipes/catalog">
2     Gnocchi
3 </erc:catalog>
```

Quellcode 6.17: Beispielcode zum Einbinden eines Ercatons in den Katalog

Der Beispiel-Code in 6.17 legt die folgenden Einträge an:

1. name: *Rezepte*, category: */*
2. name: *Nudeln*, category: */Rezepte*
3. name: *Gnocchi*, category: */Rezepte/Nudeln*

Das Ercaton mit dem `<erc:catalog>`-Tag ist Besitzer des dritten Eintrags, während den ersten beiden Einträgen lediglich ein Verweis hinzugefügt wird. Falls der entsprechende Eintrag noch nicht existiert, wird er erzeugt, ansonsten wird nur der Wert unter *RefCount* um 1 erhöht. Wird ein Ercaton gelöscht, so wird auch der entsprechende *RefCount* um 1 reduziert und die Einträge entfernt, sobald er 0 ist.

Technisch ist dies über den das Ercaton `/builtin/adder` realisiert (siehe Abschnitt 5.2.3).

6.4.1.3 Definition der Inhalte des Navigationsbaums

Das Index-Ercaton wird als Quelle in ein Ercaton `~recipes/system/catalog-def` eingebunden. Dieses dient dazu, die im Navigationsbaum anzuzeigenden Inhalte zu definieren. Ein Katalog ist dabei eine mögliche, erlaubte Quelle. Die Quellen werden getrennt abgefragt und Knoten im Navigationsbaum erzeugt. Da die Bäume zusammengefügt werden, erscheint nur jeweils ein Knoten, auch wenn mehrere Quellen die gleichen Kategorien enthalten.
Das Codebeispiel 6.19 zeigt das `~recipes/system/catalog-def` Ercaton, welches die beiden Kataloge `~recipes/catalog/laender` und `~recipes/catalog/art` eingebunden hat. Innerhalb des `<urlbase>`-Tags wird die Aktion definiert, welche bei Klick auf den entsprechenden Blattknoten im Baum ausgelöst wird. Die Aktion `/bin/catalog!list` liefert alle IDs der, per *category* definierten, Kategorie im Katalog *catalog* zurück. Mit Hilfe von `<filter>` werden nur diejenigen Ercatons im Navigationsbaum angezeigt, die dem Ercaton `/builtin/adder` gehören (siehe Abschnitt 6.4.1.2). Diese stellen die einzelnen Kategorieebenen dar. Der Filter unterdrückt „echte“ Ercatons, die sich auf gleicher Hierarchieebene befinden, da ansonsten alle Ercatons im Baum selbst auftauchen würden.

```
1  <erc:id>~recipes/system/catalog-def</erc:id>
2  <source>
3      <index>~recipes/catalog/laender</index>
4      <urlbase>/erc/atos/bin/catalog!list?catalog=~recipes/catalog/
           laender&amp;amp;category=</urlbase>
5      <urlappend/>
6      <urlencode>true</urlencode>
7      <tag>url</tag>
8      <type>category</type>
9      <filter>
10        <column>id</column>
11        <value>/builtin/adder</value>
12     </filter>
13 </source>
14 <source>
```

```
15      <index>~recipes/catalogart</index>
16      <urlbase>/erc/atos/bin/catalog!list?catalog=~recipes/catalog/art&
            amp;category=</urlbase>
17      <urlappend/>
18      <urlencode>true</urlencode>
19      <tag>url</tag>
20      <type>category</type>
21      <filter>
22          <column>id</column>
23          <value>/builtin/adder</value>
24      </filter>
25  </source>
```

Quellcode 6.18: Einbinden des Index-Ercatons in den Navigationsbaum

6.4.1.4 Integration des Katalog in die Weboberfläche

Der jetzt definierte Katalog wird anschließend in die Weboberfläche integriert. Codebeispiel 6.19 zeigt einen Ausschnitt aus dem Ercaton `~recipes/layout/nav.xhtml`, welches die Navigation für ein Webportal definiert. Hierbei generiert `activate.js` mehrere eingebettete, aktive JavaScript Applets (d.h. sie müssen nicht erst angeklickt werden). In Zeile 16 wird das Applet *com.livis.naris.applets.NavigatorApplet* eingebunden, welches das Katalog-Applet beinhaltet. Diesem werden die Parameter *configFileName*, *style* und *target* übergeben. Der Parameter *configFileName* (Zeile 17) beinhaltet den Rückgabewert der Aktion *structure* des Service-Ercatons `/bin/catalog`. Dieser wird das Ercaton `~recipes/system/catalog-def` mit der Katalog-Definition als Parameter übergeben. Die *structure*-Methode sucht in der Katalog-Definition nach Kategorie-Knoten und baut daraus einen XML-Baum für das Katalog-Applet. Per *style* in Zeile 18 ist es möglich die Farben und Schriftstile der Navigation zu bestimmen und `('target', 'list')` (Zeile 19) gibt an, dass die Kategorien als Liste angezeigt werden sollen.

```
1   <html>
2   <head>
3      <title>Special View</title>
4      <meta http-equiv="Content-Type" content="text/html;charset=utf-8"/>
5      <meta name="styled" content="true"/>
```

```
6     <erx:apply-css/>
7     <script src="/erc/aton/system/activate.js" type="text/javascript"><
          /script>
8     <erx:header xmlns="">
9        <erc:id>~recipes/layout/nav.xhtml</erc:id>
10       <erc:permission role="~any">rxb</erc:permission>
11    </erx:header>
12 </head>
13 <body>
14 ...
15 <script type="text/javascript">
16   p = new Applet("com.livis.naris.applets.NavigatorApplet", "/erc/aton
        /lib", "formslibs.jar", "100%", "100%","CatalogApplet", "0", "0",
         "middle");
17   p.addParam('configFileName', '/erc/aton/bin/catalog!structure::null?
        catalog-def=~recipes/system/catalog-def');
18   p.addParam('style', 'color:#9a0000; background-color:#d3d3d3; font-
        family:dialog,Tahoma,arial,helvetica,sans-serif; font-size:9px');
19   p.addParam('target', 'list');
20   p.endApplet();
21 </script>
22 ...
23 </body>
24 </html>
```

Quellcode 6.19: Integration des Katalogs in die Weboberfläche

6.4.2 Styles und Skins

Ercatons bestimmen ihr Aussehen über das `<erc:target name="view">` Element selbst, welches auch als *View-Target* bezeichnet wird. Fehlt dieses Element, so wird das Aussehen durch `/builtin/default.xsl`, das sogenannte *Default-Target*, festgelegt. Die Erxlib bietet jedoch eine mächtigere Version dieses *Default-Targets*. Es unterstützt einen variablen Skin und überscheibt das oben genannte *Default-Target*. Mit Hilfe eines Skins kann die grafische Oberfläche verändert werden. Ein *Stil* (Fachterm im Englischen: *Style*) ist ein Ercaton, welches einen *Skin* und eine *Startseite* beinhaltet. Eine Anwendung bietet dem Benutzer mehrere Styles an, aus denen er wählen kann. Der Vorgang des Wechsels

zwischen den Styles wird mit Hilfe der Methode *select* im Service-Ercaton `/bin/style` realisiert (`/bin/style!select`). Dies bewirkt ein Anlegen bzw. eine Änderung des Ercatons `~<user>/etc/style`, welches die Wahl des Styles für den entsprechenden Benutzer speichert. Die *Startseite* kann ein Frameset sein, das sowohl einen Bereich für die eigentlichen, variablen Ercatons, als auch einige statische Ercatons enthalten kann. Statische Ercatons sind Ercatons, die das Aussehen der Weboberfläche bestimmen und sich nicht von Benutzer zu Benutzer verändern, wie zum Beispiel die Corporate ID einer Firma. Für einen bestimmten Style müssen diese Ercatons angepasst bzw. neu entworfen werden. Der variable Bereich einer Startseite, das heißt die variablen Ercatons, wird je nach Skin-Eintrag im `~<user>/etc/style` Ercaton von `/system/standard.xsl` angepasst. Verschiedene Benutzer erhalten somit verschiedene Ansichten eines Ercatons. Der Skin-Eintrag, wie zum Beispiel *ruby* oder *standard*, spezifiert das entsprechende Skin-Ercaton (z.B. `/lib/skins/ruby/skin`). Dieses Ercaton gehört keinem bestimmten Benutzer und ist global verwendbar. Es werden darin HTML-Fragmente definiert, welche die Ansicht steuern. Außerdem werden eine CSS-Datei und Resourcen, wie JPEG oder GIF Dateien, eingebunden. Mit der Erxlib wird ein vorwiegend rot gefärbter Skin *ruby* mitgeliefert (siehe Abbildung 6.4). Ein neuer Skin, zum Beispiel *blue*, bei dem die Farbe Rot durch Blau ersetzt wird, kann durch Umfärbung der entsprechenden Bilddateien innerhalb des Skins und durch Änderung der Farbangaben in `/lib/skins/blue/skin`, sowie in der CSS-Datei, realisiert werden. Zudem muss der neue Style mit dem Skin *blue* in `~<user>/etc/style` aktiviert werden.

6.4.3 Resource-Ercatons und Referenz-Ercatons

Resource-Ercatons sind binäre Dateien, wie zum Beispiel PDF-Dateien, Bilder oder Code-Archive. Sie beinhalten weder Aktionen noch Targets und werden in keinem Index (ausser dem `/builtin/lscatalog`) gelistet. Ihre Metadaten, wie zum Beispiel die ID und Zugriffsrechte, werden direkt bei der Erzeugung definiert. Resource-Ercatons besitzen eine XML-Struktur (Document Object Model (DOM)), über die auf die Metadaten wie bei gewöhnlichen Ercatons zugegriffen werden kann, allerdings nur per Lesezugriff. Mit Hilfe des in Codebeispiel 6.20 gezeigten esh-Befehls kann ein Resource-Ercaton (hier

Bild 6.4: Der Skin 'ruby'

eine PDF-Datei „test.pdf“) erzeugt werden.

```
esh put -m text/pdf -p '~any=r' -p '~anonymous=r' -rf ./test.pdf ~
    sample/test.pdf
```

Quellcode 6.20: Erzeugung eines Resource-Ercatons mit dem esh-Befehl

Referenz-Ercatons sind Ercatons, welche eine Referenz auf ein anderes Ercaton und zusätzliche Informationen zu diesem beinhalten. Beim Zugriff über ein beliebiges Target (ausser „null“) auf ein Referenz-Ercaton, werden alle Referenzen rekursiv aufgelöst, bis schließlich ein „echtes“ Ercaton, das heißt ein Ercaton welches keine Referenz ist, erreicht wird. Das Target wird dann auf dieses Ercaton angewendet.
Ein sinnvolles Vorgehen bei Resource-Ercatons ist es, sie in Referenz-Ercatons einzubinden. Alle für das Resource-Ercaton wichtigen Informationen, wie zum Beispiel Index-Einträge (Autor, Seitenzahl, usw.), Editierfelder (z.B. Status und Freigaben) oder ein spezielles View-Target, werden, zusammen mit einem Link auf das Resource-Ercaton selbst, in einem Referenz-Ercaton gespeichert. Das Codebeispiel 6.21 zeigt ein Ercaton, welches ein Resource-Ercaton (hier eine PDF-Datei) mit der ID `~sample/test.pdf` einbindet. Das Tag `<erc:object type="erc:reference"/>` deklariert das Ercaton als Objekt vom Typ *reference* und zeigt somit der Ercato Engine, dass dieses Ercaton eine Re-

ferenz auf ein weiteres Ercaton ist. Über `<erc:id-ref>` wird eine PDF-Datei referenziert.

```
<sample>
    <erc:object type="erc:reference"/>
    <erc:id>~sample/pdfTest</erc:id>
    <erc:id-ref>~sample/test.pdf</erc:id-ref>
</sample>
```

Quellcode 6.21: Ein Ercaton mit Referenz auf eine PDF-Datei

Wird nach einem Ercaton gesucht, so wird zunächst das Referenz-Ercaton gefunden und durch dessen Aufruf die Resource, die es beinhaltet,zurückgeliefert. Der Aufruf von `https://localhost/erc/atos~sample/pdfTest` im Browser liefert also ebenso wie `https://localhost/erc/atos~sample/test.pdf` die PDF-Datei zurück.

6.5 Zusammenfassung

In diesem Kapitel wurden ausgewählte Konzepte der Ercatons-Programmierung, der Zugriffskontrolle und der Persistenz dargestellt. Des weiteren wurden Konzepte zur Anzeige, wie der Katalog, Resource-Ercatons und die Veränderung des Erscheinungsbildes von Ercatons, vorgestellt.
Hierbei ist besonders die Vererbung, die sowohl objektorientierte, als auch Thing-orientierte Aspekte unterstützt, von großer Bedeutung und wird, ebenso wie das Konzept des Katalogs und des Index, im folgenden Webprojekt verwendet werden.
Die Veränderung von Styles und Skins der Ercatons ist zwar, gerade in der Entwicklung von Webprojekten, ein zentraler Punkt, jedoch wird hierauf in dieser Studienarbeit nicht weiter eingegangen.

7 Webportal mit Ercatons

In dieser Studienarbeit wird mit Hilfe von Ercatons ein Webportal zum Zugriff auf Patientenleitlinien implementiert. Einfache Anwendungsfälle wie „Anmeldung“, „Navigation“, „Suche“, sowie das Bearbeiten von Leitlinien für bestimmte Nutzergruppen werden realisiert. Dieses Kapitel stellt zunächst die Anforderungen an das SW-System dar und geht anschließend genauer auf den Entwurf eines solchen Projektes mit Ercatons ein. Es werden wichtige Implementierungsdetails, wie das Erstellen des Grundgerüstes eines Webportals oder das Einbinden von Ercatons in den Katalog, erläutert. Abschließend folgt eine kritische Bewertung des Projektes, wobei besonders auf die Unterschiede zur traditionellen Software-Entwicklung eingegangen wird.

7.1 Anforderungsanalyse

In diesem Abschnitt werden die Anforderungen des Webportals zunächst informell beschrieben. Anschließend folgt eine Systembeschreibung mit Hilfe eines Anwendungsfalldiagramms, das das System aus Benutzersicht darstellt. Zuletzt werden die gewünschten Funktionalitäten erläutert.

7.1.1 Informelle Anforderungsbeschreibung

Die Arbeitsgemeinschaft der Wissenschaftlichen Medizinischen Fachgesellschaften (AWMF) stellt auf ihrer Webseite Leitlinien für Ärzte zur Verfügung, die ihnen bei der Entscheidungsfindung in spezifischen Situationen helfen sollen [dWMFe09]. Die Seite ist in verschiedene Themengebiete, wie zum Beispiel „Allgemeinmedizin“ oder „Gynäkologie und Geburtshilfe“ unterteilt. Diese Themengebiete wiederum besitzen Teilgebiete, wie „Allgemeine Gynäkologie“ oder „Gynäkologische Onkologie“. Jedes Teilgebiet beinhaltet eine Liste aus Leitlinieninformationen, die aus den Leitliniendokumenten selbst und

zusätzlichen Metainformationen bestehen. Abbildung 7.1 veranschaulicht dies.
Zu den Metainformationen zählen die sogenannte, eindeutige „AWMF-Register-Nummer“, die Entwicklungsstufe der Leitlinie, welche einen Wert zwischen 1 und 3 einnehmen kann, der Leitlinienname, sowie das Datum des aktuellen Standes und ein Gültigkeitsdatum. Eine Leitlinieninformation kann ein oder mehrere Leitliniendokumente (z.B. Langfassung und Kurzfassung) beinhalten, welche als PDF-Datei oder auch HTML-Webseite vorliegen können. Zusätzlich kann ein Methodenreport vorliegen, der ebenfalls Bestandteil der Leitlinieninformation ist.
Dieses Szenario soll mit Hilfe von Ercatons nachgebaut werden, wobei nicht der Inhalt der Leitlinien selbst, sondern deren Metainformationen Gegenstand der Ercatons-Modellierung sind. Die Leitliniendokumente werden über eine URL in die Ercatons eingebunden. Um den Rahmen dieser Studienarbeit nicht zu sprengen wird exemplarisch das Themengebiet „Gynäkologie und Geburtshilfe“ mit dessen Teilgebieten abgebildet. Es soll dem Benutzer zudem möglich sein, sich an dem Portal an- und abzumelden, durch die Leitlinieninformationen zu navigieren und danach zu suchen. Außerdem ist die Möglichkeit der Bearbeitung von Leitlinien für bestimmte Nutzergruppen vorgesehen. Diese Funktionalitäten werden im Abschnitt 7.1.2 als Anwendungsfall modelliert und anschließend deren Anforderungen genauer beschrieben.

Allgemeine Gynäkologie

AWMF-Reg.-Nr.			akt. Stand:	gültig bis:
003/001	Entwicklungsstufe 3	AWMF-Leitlinie **Prophylaxe der venösen Thromboembolie (VTE)** • Langfassung (PDF-Datei) • Kurzfassung (PDF-Datei)	04/2009	12/2013
029/022	Entwicklungsstufe 1	Perioperative Antibiotikaprophylaxe (Federführung: AK "Krankenhaushygiene" der AWMF)	02/2004	02/2009
015/003	Entwicklungsstufe 1	Laparoskopische Operation von Ovarialtumoren	07/2008	07/2013
015/045	Entwicklungsstufe 2	**Diagnostik und Therapie der Endometriose** • Leitlinie • Methodenreport	04/2006	04/2011
015/064	Entwicklungsstufe 1	Die laparoskopische suprazervikale Hysterektomie (LASH)	03/2008	06/2010
016/001	Entwicklungsstufe 2	**Chronischer Unterbauchschmerz der Frau** • Langfassung • Kurzfassung / Praxisleitlinie • Methodenreport	01/2009	01/2014
nvl/002	Entwicklungsstufe 3	**NVL** Asthma	12/2009	12/2013

Gynäkologische Onkologie

AWMF-Reg.-Nr.			akt. Stand:	gültig bis:
032/033	Entwicklungsstufe 2	Zervixkarzinom	01/2008	06/2010
032/034	Entwicklungsstufe 2	Endometriumkarzinom	01/2008	06/2010

Bild 7.1: Die Webseite der AWMF mit Leitlinieninformationen

7.1.2 Funktionenbeschreibung

Im Folgenden wird mit Hilfe eines Anwendungsfalldiagramms das Systemverhalten aus Anwendersicht beschrieben. Anschließend werden die für das Webportal geforderten Funktionen „Benutzerverwaltung“, „Navigation“, „Suche“, sowie das Bearbeiten von Leitlinien für bestimmte Nutzergruppen, erläutert.

7.1.2.1 Anwendungsfalldiagramm

Das Anwendungsfalldiagramm in Abbildung 7.2 stellt das Verhalten des Systems aus Anwendersicht dar. Hierbei werden die benötigten Funktionen „Anmeldung“, „Navigation“, „Suche“ und das „Bearbeiten von Leitlinien“ modelliert. Die Aktoren „Betrachter“ und „Bearbeiter“ sollen sich am System anmelden können. Beiden ist es erlaubt durch Leitlinien zu navigieren und eine Suche durchzuführen. Der Aktor „Bearbeiter“ kann zusätzlich Leitlinien bearbeiten.

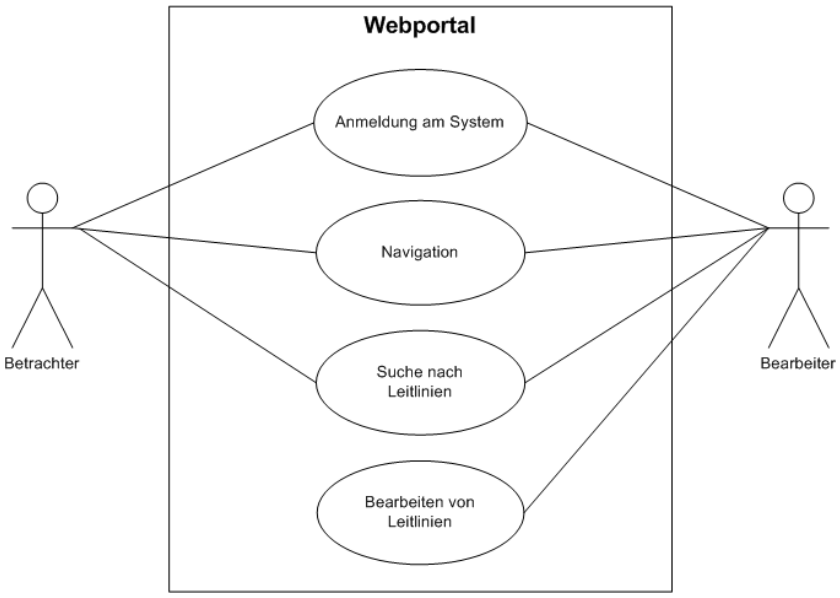


Bild 7.2: Anwendungsfalldiagramm für das Webportal

7.1.2.2 Benutzerverwaltung

Ein Nutzer (Rolle „Betrachter“ oder „Bearbeiter“) soll sich am Webportal an- und abmelden können. Hierzu wird zunächst ein Benutzer erstellt und ihm die entsprechende

Rolle zugewiesen. Er kann sich anschließend mit seinem Benutzernamen und einem entsprechenden Passwort über eine Anmeldeseite einloggen. Je nachdem welche Rolle der Benutzer zugewiesen bekommen hat, werden die für ihn nutzbaren Funktionen im Webportal angezeigt.

7.1.2.3 Navigation

Die Leitliniendokumente werden anhand ihrer Leitlinieninformation in Teilgebiete eingeordnet. Es soll möglich sein auf die Leitliniendokumente direkt, oder über ihren entsprechenden Leitlinieninformationsnamen, beispielsweise „Perioperative Antibiotikaprophylaxe“, zuzugreifen. Eine Navigation durch Teilgebiete, wie zum Beispiel „Allgemeine Gynäkologie“, soll ebenfalls implementiert werden. Die drei Gliederungspunkte der Navigation bestehen also aus „Leitliniendokument“, „Leitlinieninfo“ und „Leitlinienteilgebiet“.

7.1.2.4 Suche

Eine Suche soll nach *Leitlinieninformationsname* und *Leitlinienteilgebietname* möglich sein. Als Ergebnis wird eine Liste der entsprechenden Ercatons zurückgeliefert. Sie ist nach deren Namen aufsteigen sortiert und es werden sowohl Name, als auch die entsprechende Kategorie der Ercatons, angezeigt. Die Suche soll in die Navigation als separater Punkt mit eingebunden werden.

7.1.2.5 Bearbeiten von Leitlinien

Leitlinienteilgebiete, Leitlinieninformationen und Leitliniendokumente sind von Nutzern mit der Rolle „Bearbeiter“ bearbeitbar. Hierbei soll es möglich sein neue Leitlinienteilgebiete hinzuzufügen, vorhandene zu löschen oder sie zu ändern. Ebenso soll dies mit Leitlinieninformationen und Leitliniendokumenten möglich sein.

7.2 Entwurf

Ein SW-System wird mit Ercatons nicht, wie in der traditionellen Software-Entwicklung, komplett im Voraus modelliert, sondern Stück für Stück aufgebaut. Es wächst also anstatt modelliert zu werden und es ist aus diesem Grunde nicht nötig ein Klassendiagramm

oder ähnliches zu erstellen. Es genügen Beschreibungen und Vergleiche der Ercatons. Im Folgenden wird deshalb nur auf eine grobe Abbildung der Leitlinieninformationen auf Ercatons eingegangen und hierbei initial benötigte Attribute und Funktionen beschrieben. Anschließend folgt eine genaue Beschreibung der Funktionen „Anmeldung“, „Navigation“ und „Suche“.

7.2.1 Abbildung der Leitlinieninformationen auf Ercatons

Die Leitlinieninformationen werden folgendermaßen auf Ercatons abgebildet: Für jedes Leitlinienteilgebiet, wie zum Beispiel „Allgemeine Gynäkologie“ oder „Gynäkologische Onkologie“, wird ein Ercaton erstellt, das sogenannte *Leitlinienteilgebiet*-Ercaton. Es beinhaltet eine Liste der zugehörigen Leitlinien. Jede Zeile dieser Liste ist erneut ein Ercaton, das *BibliografischeLeitlinienInfo*-Ercaton. Es beinhaltet sowohl Metainformationen zu den Leitlinien, als auch eine Liste von Leitliniendokumenten selbst. Diese Leitliniendokumente werden auf Ercatons vom Prototyp *Leitliniendokument* abgebildet und besitzen nicht das Dokument selbst, sondern einen Namen und Link auf dieses. Im Folgenden werden die einzelnen Prototyp-Ercatons genauer beschrieben und hierbei auf ihre Attribute und Aktionen eingegangen.

7.2.1.1 Leitlinienteilgebiet-Ercaton

Der *Leitlinienteilgebiet* Prototyp beinhaltet den Namen des Leitlinienteilgebiets und eine Liste zugehöriger *BibliografischeLeitlinienInfo*-Ercatons. Es sollen die Aktionen „Bearbeiten“, „Löschen“ und „Neues Leitlinienteilgebiet anlegen“ implementiert werden. Ein *Leitlinienteilgebiet*-Ercaton besitzt also die nachfolgend aufgeführten Attribute:

- **Teilgebietname**: Jedes *Leitlinienteilgebiet*-Ercaton hat genau einen, eindeutigen Teilgebietnamen, zum Beispiel „Allgemeine Gynäkologie“.
- **Leitlinienliste**: Die Leitlinienliste wird als mehrwertiges Attribut modelliert und beinhaltet eine Menge an Links auf *BibliografischeLeitlinienInfo*-Ercatons, sowie deren Namen. Es soll möglich sein die Liste beliebig zu erweitern oder zu reduzieren.
- **Ercaton-ID**: Für den Prototyp wird die ID
  `~webportal/base/leitlinienteilgebiet`
  verwendet. Die Beispiele werden mit einer ID der Form

`~webportal/sample/<Teilgebietname>`
gebildet.

7.2.1.2 BibliografischeLeitlinienInfo-Ercaton

In einem *BibliografischeLeitlinienInfo*-Ercaton werden die Metainformationen zu den Leitliniendokumenten gespeichert, sowie eine Liste von *Leitliniendokument*-Ercatons. Außerdem sollen die Aktionen „Bearbeiten“, „Löschen“ und „Neue Leitlinieninfo anlegen“ hinzugefügt werden.
Folgende Attribute werden benötigt:

- **Leitlinienname**: Jede Leitlinie besitzt genau einen, eindeutigen Leitliniennamen, zum Beispiel „Diagnostik und Therapie der Endometriose“ oder „Perioperative Antibiotikaprophylaxe“.
- **AWMF-Reg.-Nr.**: Die „AWMF-Reg.-Nr.“ ist eine von der AWMF vergebene, eindeutige Registernummer für Leitlinien. Sie ist von der Form „003/001“ oder auch „nvl/002“, beinhaltet also sowohl numerische als auch alphanumerische Zeichen und wird deshalb als Typ „string“ modelliert.
- **Entwicklungsstufe**: Eine Leitlinie hat genau eine der Entwicklungsstufen 1, 2 oder 3. Sie kann sich im Laufe der Zeit, je nach Erweiterung der Leitliniendokumente, verändern. Es bietet sich an dies als Auswahl-Menü zu realisieren.
- **aktueller Stand**: Jede Leitlinie hat einen aktuellen Stand der Form „MM/JJJJ“.
- **gültig bis**: Das Ablaufdatum einer Leitlinie ist ebenso wie der aktuelle Stand von der Form „MM/JJJJ“ und wird als „date“ modelliert.
- **Leitliniendokumentliste**: Die Leitliniendokumentliste wird als mehrwertiges Attribut realisiert. Sie fasst eine unbestimmte Menge Links auf *Leitliniendokument*-Ercatons, sowie die zugehörigen Namen. Die Liste soll beliebig erweiterbar oder reduzierbar sein.
- **Ercaton-ID**: Der Prototyp besitzt die ID `~webportal/base/bibliografischeLeitlinienInfo`.
  Für die Beispiele bieten sich sowohl der Leitlinienname, als auch die AWMF-Reg.-Nr. an, da beide eindeutig sind. Allerdings beinhaltet die AWMF-Reg.-Nr. das

Sonderzeichen „/“, welches in einer Ercaton-ID nicht darstellbar ist. Deshalb fiel die Entscheidung auf eine ID der Form
`~webportal/sample/<Teilgebietname>/<Leitlinienname>`.

7.2.1.3 Leitliniendokument-Ercaton

Der *Leitliniendokument* Prototyp beinhaltet nicht das Leitliniendokument selbst, sondern nur dessen Namen und einen Link auf seine Web-Quelle. Es existieren sowohl Dokumente vom Typ „Methodenreport“, als auch „Leitlinie“. Zudem gibt es „Kurzfassungen“ und „Langfassungen“ einer Leitlinie. Die benötigten Attribute lauten somit:

- **Dokumentname**: Der Dokumentname ist entweder „Methodenreport", „Leitlinie“, „Kurzfassungen“ oder „Langfassung“ und wird als Auswahl-Menü realisiert. Er ist demnach nicht eindeutig.
- **Link**: Ein weiteres Attribut ist der Link auf die Webquelle des Leitliniendokuments.
- **Ercaton-ID**: Für das Prototype-Ercaton wird die ID
  `~webportal/base/leitliniendokument/`
  verwendet. Die Beispiele werden mit einer ID der Form
  `~webportal/sample/<Teilgebietname>/<Leitlinienname>/<Typ>`
  gebildet. Der *Typ* kann hierbei „Leitlinie“, „Methodenreport“ oder, falls eine Lang- und Kurzfassung der Leitlinie existiert, „Kurzfassung“ bzw. „Langfassung“ sein.

Leitliniendokument-Ercatons sollen kopiert, bearbeitet und gelöscht werden können.

7.2.2 Detaillierte Funktionenbeschreibung

Nachfolgend werden die Funktionen „Benutzerverwaltung“ und „Navigation“ detailliert beschrieben. Hierbei wird besonders auf die benötigten Ercatons und die darunter liegenden Konzepte eingegangen.

7.2.2.1 Benutzerverwaltung

Die Benutzerverwaltung basiert auf dem Zugriffsmodell der Ercatons (6.2). Benutzer können sich am Webportal an- und abmelden. Es können Benutzer hinzugefügt und ihnen verschiedene Rollen mit entsprechenden Rechten zugewiesen werden. Es werden die folgenden Rollen erstellt:

- **Bearbeiter**: Mitglieder der Rolle `~bearbeiter` sollen neue Benutzer hinzufügen und Leitlinie erstellen, editieren, löschen und danach suchen können.
- **Betrachter**: Ein Betrachter soll Leitlinien nur lesen und danach suchen können. Er hat keine Rechte sie zu bearbeiten. Das zugehörige Role-Ercaton lautet `~betrachter`.

Zu Testzwecken sollen die nachfolgend aufgelisteten Benutzer erzeugt und ihnen die entsprechenden Rollen zugewiesen werden:

- **Klaus Testmann**: Dieser Benutzer wird im Ercaton `~klaus` realisiert und besitzt die Rolle „Bearbeiter“.
- **Susi Testfrau**: Ihr soll die Rolle „Betrachter“ zugewiesen werden. Das entsprechende User-Ercaton lautet `~susi`.

7.2.2.2 Navigation

Die Navigation wird mit Hilfe des Katalog-Konzepts (siehe 6.4.1) realisiert. Hierbei werden die Ercatons in die Kategorien „Leitlinienteilgebiet“, „Leitlinieninfo“ und „Leitliniendokument“ eingebunden. Es soll sowohl die Kategorie, als auch der Name des entsprechenden Ercatons angezeigt werden. Das Index-Ercaton für den Katalog soll die ID `~webportal/catalog` besitzen.

7.2.2.3 Suche

Die Suche wird mit Hilfe des Index (6.3.2) implementiert. Es soll möglich sein nach Leitliniennamen und Leitlinienteilgebieten zu suchen. Die Suche soll unter dem Namen `Suchabfragen\Leitliniensuche` in die Navigation eingebunden werden. Die Ergebnisliste muss sowohl den Namen der entsprechenden Leitlinienteilgebiete und Leitlinieninfos, als auch die Kategorie in der sie im Katalog eingebunden sind, anzeigen. Es wird ein Query-Ercaton benötigt und hierfür die Ercato-ID `~webportal/query/leitliniensuche` gewählt. Das Query-Ercaton beinhaltet die Attribute „name“ und „category“ und sortiert die Ercatons aufsteigen nach ihrem Namen.

7.3 Implementierung

In diesem Abschnitt wird auf wichtige Implementierungsdetails des Webportals eingegangen. Neben dem Grundgerüst wird die Erstellung eines Auswahl-Menüs für die Entwicklungsstufen und das Einbinden mehrerer Leitliniendokumente in eine Liste erläutert. Des Weiteren wird erklärt wie die Navigation erstellt wurde und die Benutzerverwaltung für dieses Projekt aufgebaut ist. Als Abschluss folgt eine Beschreibung der Suchfunktion.

7.3.1 Webportal Grundgerüst

Zunächst wird das Grundgerüst für das Webportal selbst erstellt, in das später per Katalog die einzelnen Leitfäden-Ercatons eingebunden werden. Hierzu ist zunächst ein Start-Ercaton `~webportal/start` nötig, welches das HTML-Grundgerüst der Webseite beinhaltet und den Style *Ruby* einbindet (vgl. 6.4.2). Dieses Ercaton wird für den Einstieg in das Webportal verwendet und per `https://localhost/erc/atos/~webportal/start` im Webbrowser aufgerufen. Je nach Belieben kann der HTML-Code der Webseite in verschiedene Ercatons ausgegliedert werden, wie dies auch bei gewöhnlichen Webseiten möglich ist. Zum Beispiel wäre ein `~webportal/layout/nav.xhtml` Ercaton denkbar, welches die Navigation beinhaltet, und ein `~webportal/layout/top.xhtml` Ercaton für den Kopf der Webseite.
Mittels des Installations-Skripts *installApp.sh*, welches automatisch mit der Ercato Engine installiert wurde, wird das Grundgerüst des Webportals deployed. Es müssen hierbei der Port, der Host, das `~root`-Passwort und der Projektname des Webportals angegeben werden. Das Codebeispiel 7.1 zeigt den entsprechenden Aufruf. Das Skript liegt im Verzeichnis `\erc.install\erc.pckgs\prj\script` und überprüft die Eingaben, meldet den Benutzer `~root` mit dem entsprechenden Passwort per ESH-Befehl an der Engine an und ruft die Java-Klasse `com.ercato.lib.util.client.Install` mit den entsprechenden Argumenten auf. Eine Dokumentation zur Verwendung dieser Klasse befindet sich im Anhang unter A.3.

```
env WEBPORTAL_PORT="80" WEBPORTAL_HOST="localhost" ROOT_PASSWORD="6
    h98xLW" ./installApp.sh -prj webportal -c  update
```

Quellcode 7.1: Installation des Webportals

7.3.2 Auswahl-Menü für die Entwicklungsstufe

Für die Auswahl einer der drei Entwicklungsstufen soll ein Auswahl-Menü erzeugt werden. Es wird in das *BibliografischeLeitlinienInfo*-Ercaton eingebunden und beinhaltet die Einträge „Entwicklungsstufe 1“, „Entwicklungsstufe 2“ und „Entwicklungsstufe 3“. Hierzu ist es nötig ein Listen-Ercaton zu erstellen, welches die Einträge des Auswahl-Menüs beinhaltet. Codebeispiel 7.2 zeigt das Listen-Ercaton `~webportal/values/entwicklungsstufe`. Das `<content>`-Tag beinhaltet den tatsächlichen Wert, während `<display>` den anzuzeigenden Text speichert.

```
<value-list>
  <erc:id>~webportal/values/entwicklungsstufe</erc:id>
  <name></name>
  <usevalue>true</usevalue>
  <values>
    <value>
      <content>1</content>
      <display>Entwicklungsstufe 1</display>
    </value>
    <value>
      <content>2</content>
      <display>Entwicklungsstufe 2</display>
    </value>
    <value>
      <content>3</content>
      <display>Entwicklungsstufe 3</display>
    </value>
  </values>
  <erx:actions>
    <erc:target name="list">/system/values2list.xsl</erc:target>
  </erx:actions>
</value-list>
```

Quellcode 7.2: Ein Listen-Ercaton mit Einträgen für die Entwicklungsstufe

Das Codebeispiel 7.3 zeigt wie Constraint und Layout für das Auswahl-Menü im *BibliografischeLeitlinienInfo*-Ercaton realisiert wurden. Für das Feld der Entwicklungs-

stufe wird im `<layout>`-Tag (Zeile 5), auf das per *lr*-Attribut (Zeile 8) verwiesen wird, die Größe definiert und ihm die Eigenschaft *select-dropdown* zugewiesen. Diese stellt das Feld als Auswahl-Menü dar und ist im `/system/standard.xsl` Ercaton definiert. Mittels eines Constraint, der durch das *cr*-Attribut (Zeile 8) referenziert wird, wird das oben genannte Listen-Ercaton als Quelle für das Auswahl-Menü angegeben. Die grafische Darstellung dieses Auswahl-Menüs wird in Abbildung 7.3 gezeigt.

```
1 <erx:constraints>
2    <erx:constraint erc:name="entwicklungsstufe" source-type="value-
        list" source="~webportal/values/entwicklungsstufe" />
3 </erx:constraints>
4 <erx:layouts>
5    <erx:layout erc:name="select" method="select-dropdown" size="20"/>
6 </erx:layouts>
7 <erx:fields>
8    <erx:field erc:name="entwicklungsstufe" field-name="
        Entwicklungsstufe" lr="select" cr="entwicklungsstufe"/>
9 </erx:fields>
```

Quellcode 7.3: Constraint und Layout für das Auswahl-Menü

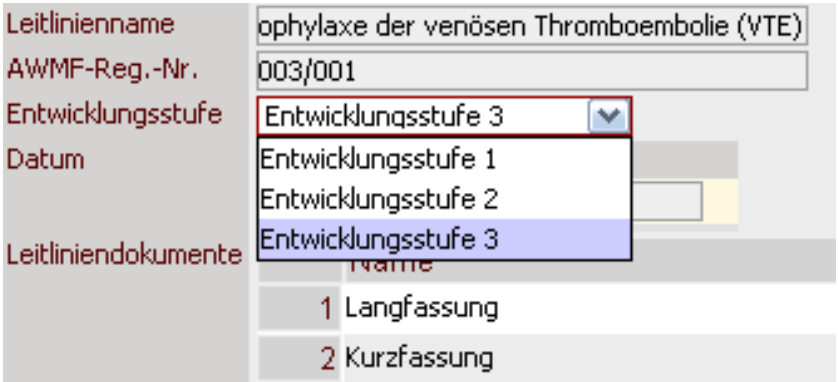


Bild 7.3: Anzeige des Auswahl-Menüs in einem BibliografischeLeitlinienInfo-Ercaton

7.3.3 Mehrwertige Attribute

Das Ercaton *BibliografischeLeitlinienInfo* soll eine unbestimmte Menge an *Leitliniendokument*-Ercatons in einer Leitliniendokument-Liste erfassen. Ebenso besitzt das

Leitlinienteilgebiet-Ercaton eine Liste aus zugehörigen *BibliografischeLeitlinienInfo*-Ercatons. Beide Listen werden analog modelliert, weshalb in diesem Abschnitt nur auf die Leitliniendokument-Liste im *BibliografischeLeitlinienInfo*-Ercaton näher eingegangen wird.

Die Liste wird mit Hilfe von *fields* (vgl. 6.1.2) erstellt und ist in Codebeispiel 7.4 ausschnittsweise dargestellt. Hierbei wird der Link auf das *Leitliniendokument*-Ercaton mittels des `<id-ref>`-Tags eingebunden. Dieser beinhaltet eine Ercaton-ID und zeigt der Ercato Engine, dass es sich um eine Referenz auf ein weiteres Ercaton handelt. Durch Klick auf einen derartigen Link öffnet sich das entsprechende *Leitliniendokument*-Ercaton.

```
<columns erx:list-ref="leitliniendokumente" erx:nodename="column">
    <column erc:inherited-as="vector">
    <dokumentname erx:field-ref="string" erx:field-name="Name"/>
    <dokumentlink>
        <erc:id-ref erx:field-ref="idref" erx:field-name="Link"/>
    </dokumentlink>
  </column>
</columns>
```

Quellcode 7.4: Liste für Leitliniendokumente mit Hilfe des Table-Control

Abbildung 7.4 zeigt die Leitliniendokument-Liste in einem Beispiel-Ercaton. Die Symbole für das Hinzufügen, Löschen und Verschieben von Tabellen-Zeilen sind in der *erxlib* definiert und werden automatisch hinzugefügt.

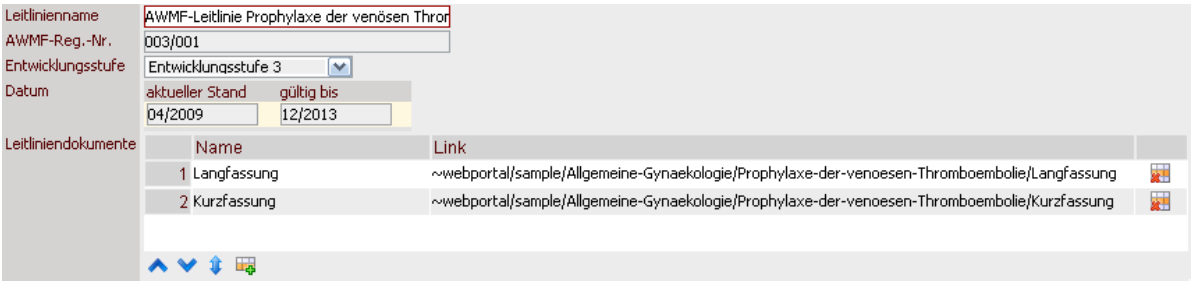


Bild 7.4: Eine Leitliniendokument-Liste

7.3.4 Bearbeiten von Leitlinien

Die Aktionen „Bearbeiten“, „Löschen“ und „Neu Anlegen“ von Leitlinien werden den jeweiligen Ercatons mittels der entsprechenden Service-Ercatons hinzugefügt. Codeausschnitt 7.5 zeigt dies. Für die Aktion „Bearbeiten“ wird die *main*-Funktion im Service-Ercaton `/bin/edit` aufgerufen. Es stellt einen Mechanismus zum Bearbeiten von Ercatons zur Verfügung. Entsprechend wird für das „Löschen“ von Ercatons das Service-Ercaton `/bin/rm` angesprochen. Die Methode *wizard* bietet hier zusätzlich eine Webseite mit einer Nachfrage, ob das entsprechende Ercaton wirklich gelöscht werden soll. Die Funktion "Neue Leitlinieninfo anlegen"wird durch `/bin/cp!copyAndEdit` realisiert. Sie kopiert das Ercaton und wechselt in den „Bearbeiten“-Modus. Um die Felder leer anzuzeigen, muss ihnen jeweils das Attribut `erx:clear-on-copy="true"` hinzugefügt werden. Es löscht den Inhalt der Felder beim Kopieren des Ercatons. Das Aussehen der Schaltflächen wird durch das Attribut *erx:group="toolbar"* und das *View*-Target bestimmt (vgl. 6.4.2).

```
1 <erx:actions>
2   <erc:action erx:field-name="Bearbeiten" default-target="view"
       erx:group="toolbar" name="edit">/bin/edit</erc:action>
3   <erc:action erx:field-name="Löschen" erx:group="toolbar" name="
       remove">/bin/rm!wizard</erc:action>
4   <erc:action erx:field-name="Neue Leitlinieninfo anlegen" erx:group="
       toolbar" name="copy">/bin/cp!copyAndEdit</erc:action>
5 </erx:actions>
```

Quellcode 7.5: Hinzufügen der Aktionen „Bearbeiten“, „Löschen“ und "Neue Leitlinieninfo anlegen"

Abbildung 7.5 zeigt ein Beispiel für ein vollständiges *BibliografischeLeitlinienInfo*-Ercaton.

7.3.5 Realisierung der Navigation

Die Navigation wird mit Hilfe des Katalog-Konzepts realisiert (siehe 6.4.1). Die drei Gliederungspunkte der Navigation sollen aus „Leitliniendokument“, „Leitlinieninfo“ und „Leitlinienteilgebiet“ bestehen. Das dafür benötigte Index-Ercaton `~webportal/catalog`

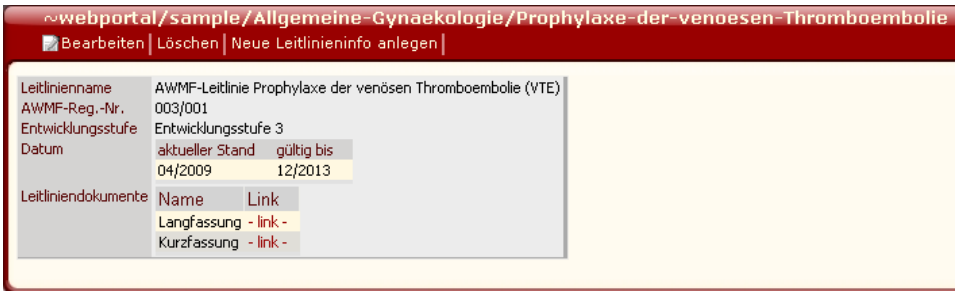


Bild 7.5: Beispiel eines BibliografischeLeitlinienInfo-Ercatons

wird in Codebeispiel 7.6 gezeigt. Um die entsprechenden Kategorien in den Navigationsbaum einzubinden muss der Katalog im Ercaton `~webportal/system/catalog-def` als Quelle definiert werden.

```
<index-table>
  <erc:type>index</erc:type>
  <erc:id>~webportal/catalog</erc:id>
  <erx:permissions>
    <erc:permission name="~any">rxb</erc:permission>
  </erx:permissions>
  <erc:index>
    <erc:name erx:display="Kategorie">category</erc:name>
  </erc:index>
  <erc:index>
    <erc:name erx:display="Name">name</erc:name>
  </erc:index>
  <erc:index>
    <erc:name>RefCount</erc:name>
    <erc:index-type>int</erc:index-type>
    <erc:adder/>
  </erc:index>
</index-table>
```

Quellcode 7.6: Index-Ercaton für Katalog

Die *BibliografischeLeitlinienInfo*-Ercatons werden anhand des `<catalog>`-Tags mit den Attributen `category="/Leitlinien/Leitlinieninfo"` und `id-ref="~webportal/catalog"` in den Katalog eingebunden. Die *Leitlinienteilgebiet*- und *Leitliniendokument*-Ercatons werden entsprechend integriert, jedoch in den

Kategorien „/Leitlinien/Leitlinienteilgebiet“ bzw. „/Leitlinien/Leitliniendokument“. Der jetzt definierte Katalog wird anschließend in die Weboberfläche integriert (siehe 6.4.1.4).

7.3.6 Benutzerverwaltung

Die Benutzerverwaltung basiert auf dem Zugriffsmodell der Ercatons (siehe 6.2). Es werden die Benutzer „Susi Testfrau“ und „Klaus Testmann“ erstellt. Hierzu werden die Ercaton-IDs `~susi` und `~klaus` verwendet. Die Rolle `~admin` erhält das Recht die User-Ercatons zu verändern. Codebeispiel 7.7 zeigt das User-Ercaton für Susi Testfrau.

```
<user>
   <erc:id>~susi</erc:id>
   <erc:type>user</erc:type>
   <erc:permission role="~admin">rwb</erc:permission>
   <person>
      <first-name>Susi</first-name>
      <last-name>Testfrau</last-name>
   </person>
</user>
```

Quellcode 7.7: User-Ercaton für Susi Testfrau

Außerdem werden die Rollen „Betrachter“ (`~webportal/roles/betrachter`) und „Bearbeiter“ (`~webportal/roles/bearbeiter`) erzeugt. Codebeispiel 7.8 zeigt die Rolle `~webportal/roles/betrachter`. Sie wird sowohl „Susi“, als auch „Klaus“ zugewiesen. Die Rolle `~webportal/roles/bearbeiter` ist analog aufgebaut, allerdings enthält diese nur eine Zuweisung zu `~klaus`.

Um die Aktionen „Bearbeiten“, „Löschen“ und „Neue Leitlinieninfo anlegen“ nur der Rolle „Bearbeiter“ und somit dem Benutzer „Klaus Testmann“ zugängig zu machen, werden die beiden, in 7.9 gezeigten, Permission-Tags in die Prototyp-Ercatons *Leitlinienteilgebiet*, *BibliografischeLeitlinienInfo* und *Leitliniendokument* eingetragen. Hierbei wird der Rolle „Betrachter“ kein Schreibrecht gewährt und er darf keine Aktionen ausführen.

Damit sich die Benutzer mit ihrem jeweiligen Benutzernamen und Passwort am Webportal anmelden können, müssen sie zunächst mit Hilfe des sogenannten *User-*

```
<role>
  <erc:id>~webportal/roles/betrachter</erc:id>
  <erc:type>role</erc:type>
  <name>Betrachter</name>
  <erc:role>
    <erc:granted-for role="~susi"/>
    <erc:granted-for role="~klaus"/>
  </erc:role>
</role>
```

Quellcode 7.8: Role-Ercaton für die Rolle „Betrachter“

```
<permissions>
    <erc:permission name="~webportal/roles/betrachter">rb</
       erc:permission>
    <erc:permission name="~webportal/roles/bearbeiter">rxwb</
       erc:permission>
</permissions>
```

Quellcode 7.9: Permission-Tags

Manager, der bei der Installation der Ercato Engine mitgeliefert wurde, erstellt und somit der Ercato Engine bekannt gemacht werden. Der User-Manager lässt sich über eine Weboberfläche unter der Adresse `https://localhost/erc/atos/sbin/usrmgr` steuern. Zum Erstellen und Löschen von Benutzern werden `~root` Rechte benötigt.
Für das Webportal wurde der Benutzer „susi“ mit dem Passwort „susipw“, sowie „klaus“ mit Passwort „klauspw“ erstellt. Beide können sich jetzt am Webportal anmelden. „Susi Testfrau“ kann jedoch Ercatons nicht verändern, da ihr die entsprechenden Rechte fehlen. Der Versuch die „Bearbeiten“ Aktion aufzurufen wirft eine Fehlermeldung, wie in Abbildung 7.6 dargestellt.

7.3.7 Suchfunktion

Um eine einfache Suchfunktion nach Leitliniennamen und Teilgebietnamen zu realisieren, genügt es ein Query-Ercaton, wie in Codebeispiel 7.10 gezeigt, zu erstellen. Hierbei wird der Katalog `~webportal/catalog` als Index verwendet. Für komplexere Abfragen ist es jedoch auch möglich beliebige Index-Ercatons einzubinden, welche die entsprechenden Attribute beinhalten. Es werden die Spalten (`<column>`-Tag) „name“ und „category“

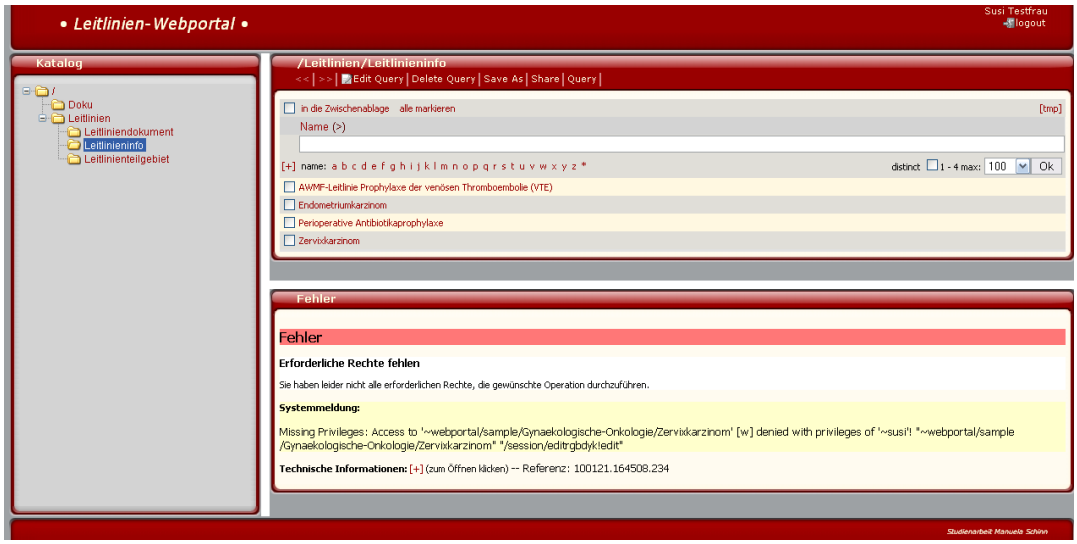


Bild 7.6: Fehlermeldung bei nicht erlaubtem Zugriff auf die „Bearbeiten“ Aktion

angezeigt, wobei die Ergebnisse aufsteigend nach Namen sortiert sind. Dies wird durch `<sort>asc</sort>` erreicht. Außerdem ist eine Filterung nach Anfangsbuchstaben des Namens möglich (`<firstletter>true</firstletter>`). Realisiert wird diese Funktion mit Hilfe des `/lib/base/query`-Ercatons, das der Abfrage als Prototyp dient. Es enthält außerdem das Grundgerüst für Suchabfragen und ruft benötigte Funktionen im Service-Ercaton `/bin/query` auf. Das Ercaton `/system/values2list.xsl` sorgt dafür, dass die resultierenden Ercatons als Liste zurückgeliefert werden. In Abbildung 7.7 ist das Ergebnis der Suche dargestellt. Es ist möglich Query-Ercatons über die Weboberfläche zu verändern, zu löschen oder neue zu erstellen. Dieser Mechanismus ist bereits in der Erxlib implementiert und muss nicht zusätzlich hinzugefügt werden.

```
1  <query>
2    <erc:clone>/lib/base/query</erc:clone>
3    <erc:id>~webportal/query/leitliniensuche</erc:id>
4    <erc:permission name="~any"/>
5    <name>/Suchabfragen/Leitliniensuche</name>
6    <description>Suche nach Leitlinieninfos</description>
7    <query>
8      <index>
9        <erc:id-ref>~webportal/catalog</erc:id-ref>
10     </index>
```

```
    <columns>
      <column>
        <name>name</name>
        <sort>asc</sort>
        <width/>
        <idrefcol>id</idrefcol>
      </column>
      <column>
        <name>category</name>
      </column>
    </columns>
    <filters>
      <or>
        <and>
          <column>name</column>
          <firstletter>true</firstletter>
        </and>
      </or>
    </filters>
    <pagesize erx:field-ref="pagesize">200</pagesize>
  </query>
  <catalog>
    <erc:catalog category="/Suchabfragen/"/>
  </catalog>
  <erx:actions>
    <erc:target name="list">/system/values2list.xsl</erc:target>
  </erx:actions>
</query>
```

Quellcode 7.10: Query-Ercaton

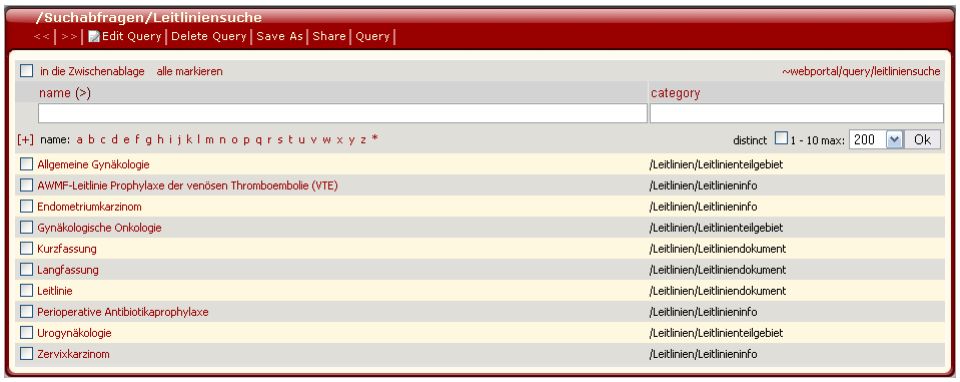


Bild 7.7: Beispiel-Suchabfrage

7.4 Bewertung

In diesem Abschnitt folgt eine kritische Bewertung der Implementierung dieses Webportals mit Ercatons. Hierbei wird besonders auf Unterschiede zur traditionellen Software-Entwicklung eingegangen und die Vor- und Nachteile einer Ercatons-Entwicklung herausgearbeitet.

7.4.1 Vorteile

Die Entwicklung des Webportals für Patientenleitfäden mit Hilfe von Ercatons hat einige Vorteile gegenüber der traditionellen Software-Entwicklung aufgezeigt. Sie werden im Folgenden beschrieben:
Die Möglichkeit ein einzelnes Ercaton, unabhängig von anderen Ercatons, zu verändern hat sich als besonders hilfreich erwiesen. Zum Beispiel war es nötig dem Ercaton für die Leitlinieninformation „Perioperative Antibiotikaprophylaxe“ im Nachhinein einen Kommentar hinzuzufügen. Dies hätte man zwar im Voraus erkennen und modellieren können, da jedoch der größte Teil der *BibliografischeLeitlinienInfo*-Ercatons kein Kommentarfeld benötigte, wurde es übersehen. Wäre dies in einer Klassen-basierten Programmiersprache aufgetreten, so hätte die Klasse, von der das entsprechende Objekt instanziiert wurde, verändert und das komplette Projekt neu kompiliert und getestet werden müssen. Dies wäre nicht nur ein großer Aufwand, sondern auch nicht von Vorteil. Da ein Kommentarfeld in nur sehr wenigen *BibliografischeLeitlinienInfo*-Ercatons benötigt wird, macht es keinen Sinn es in allen vorzusehen. Durch Hinzufügen eines `<kommentar>`-Tags mit

entsprechendem Inhalt zu dem „Perioperative Antibiotikaprophylaxe“-Ercaton konnte das Problem hier schnell und sauber gelöst werden.
Wird jedoch im Nachhinein festgestellt, dass eine Gruppe an Ercatons, die vom selben Prototyp abstammen, eine zusätzliche Eigenschaft benötigen, so genügt es den Prototypen zu verändern. Alle Ercatons, die von ihm geerbt haben, werden diese zusätzliche Eigenschaft ebenfalls erhalten. Zum Beispiel wurde bei diesem Projekt das Zugriffsmodell erst im Nachhinein implementiert. Hierzu wurden Role-Ercatons und User-Ercatons erstellt und in die Engine eingefügt. Den Prototypen der entsprechenden Ercatons wurde ein `<permission>`-Tag hinzugefügt. Alle davon abhängigen Ercatons waren dadurch ebenfalls mit den entsprechenden Zugriffsrechten behaftet. Im Gegensatz zu vielen Programmiersprachen besitzen Ercatons ein sehr einfach zu implementierendes Zugriffsmodell.
Wie die beiden oben genannten Beispiele zeigen, war es nicht nötig das Projekt komplett im Voraus zu planen. Es hat sich vielmehr nach und nach entwickelt.

7.4.2 Nachteile

Ein Nachteil der Entwicklung dieses Projektes mit Ercatons ist die große Anzahl an Klicks, die im Vergleich zur Originalstruktur in Abbildung 7.1 benötigt werden. Der Grund hierfür ist die Dekomposition in verschiedene Ercatonstufen. Außerdem ist es bei derartigen Projekten häufig notwendig die Anzeige der Weboberfläche anzupassen. Es müssen Skin und Style verändert und neue Grafiken erstellt werden (siehe 6.4.2). Der Aufwand um dies mit Ercatons zu realisieren ist jedoch unklar. Ein weiterer großer Nachteil ist der Arbeitsaufwand, der nötig ist um die Ercato Engine, den Java EE-Anwendungsserver, die Datenbank und diverse nötige Werkzeuge, wie Eclipse und Cygwin, zu installieren. Hinzu kommt die Einarbeitungszeit in diese völlig neue Art der Denkweise und die noch sehr spärliche Dokumentation. Es muss der Quellcode vieler Builtin- und Service-Ercatons untersucht werden um die Konzepte zu verstehen.

7.5 Zusammenfassung

In diesem Abschnitt wurde die Entwicklung eines Webportals mit Hilfe von Ercatons beschrieben. Zunächst erfolgte eine Anforderungsanalyse und der Entwurf des Portals,

wobei die Abbildung der Patientenleitfäden auf Ercatons, sowie die benötigten Funktionalitäten, beschrieben wurden. Anschließend wurde die Implementierung genauer erläutert. Hierzu zählten unter Anderem die Erstellung des Grundgerüstes für das Webportal, die Abbildung von mehrwertigen Attributen und die Realisierung der Benutzerverwaltung. Als Abschluss des Kapitels wurde die Realisierung dieses Webportals mit Ercatons kritisch bewertet und gegen die traditionelle Software-Entwicklung abgegrenzt.

8 Epilog

In diesem Kapitel werden zunächst die Kernaussagen der Studienarbeit zusammengefasst und anschließend ein Ausblick auf zukünftige Arbeiten und offene Punkte gegeben. Ein Resume, in dem die Vor- und Nachteile der Entwicklung mit Ercatons zusammengefasst werden, rundet diese Studienarbeit ab.

8.1 Zusammenfassung

Diese Studienarbeit hat eine Einführung in die Programmierumgebung der Ercatons gegeben und dabei das Programmiermodell der organische Programmierung beschrieben. Zunächst wurde die Prototyp-basierte Programmierung definiert und hierbei eine Einführung in die prototyp-basierten Sprachen Self und JavaScript gegeben. Es wurde der Aufbau eines *Things* gezeigt und mit dessen Hilfe die Idee der organische Programmierung erläutert, sowie gegen die Prototyp-basierte Programmierung abgegrenzt. Als technische Infrastruktur hierzu wurden die Ercatons eingeführt und hierbei die verschiedenen Typen der Ercatons, das Programmiermodell und die sogenannte *Ercato Markup* erklärt. Anschließend folgte eine Beschreibung der Ercato Engine, wobei sich gezeigt hat, dass Service-Ercatons und Builtin-Ercatons, die bereits bei der Installation der Engine mitgeliefert werden, nützliche vordefinierte Funktionen beinhalten und das Arbeiten mit Ercatons erleichtern. Die verschiedenen Zugriffsarten auf Ercatons und die Möglichkeit mit Hilfe der *Ercato Shell* Ercatons zu löschen oder in die Engine einzufügen, rundeten das Kapitel der Ercatons Grundlagen ab.
Anschließend folgte eine Beschreibung der Realisierung verschiedener Konzepte des Software-Entwurfs mit Ercatons. Mit Hilfe des *XOperators* ist eine Vererbung definiert, die auf einer Addition beruht. Dies wurde anhand eines objektorierten und eines Thing-orientierten Beispiels erläutert. Das Zugriffsmodell der Ercatons kann sowohl Rollen-basierte, als auch die, aus der OOP bekannte, Paket- oder Modul-basierte Zu-

griffskontrolle realisieren. Die Persistenz von Ercatons ist auf einem Transaktions-Layer realisiert, welcher direkt auf dem Filesystem aufsetzt. Mit Hilfe des Konzepts des Index ist es möglich Suchabfragen zu formulieren. Auch mehrwertige Attribute können mit den sogenannten *fields* problemlos abgebildet werden. Zudem wurden einige Konzepte zu Anzeige, wie der Katalog und Resource-Ercatons, beschrieben. Der Katalog ist hierbei ein nützliches Mittel um Ercatons in ein Webportal einzubinden und eine Navigation zu realisieren. Des Weiteren wurde gezeigt, wie es möglich ist das Aussehen von Ercatons zu verändern.
Anschließend wurde ein Webportal zum Zugriff auf Patientenleitfäden mit Hilfe von Ercatons realisiert. Hierbei wurde zunächst eine Anforderungsanalyse erstellt und im Entwurf die Abbildung der Leitfäden auf Ercatons sowie die gewünschten Funktionalitäten beschrieben. Im Abschnitt zur Implementierung wurde auf wichtige Implementierungsdetails des Webportals eingegangen. Neben dem Grundgerüst wurde die Erstellung eines Auswahl-Menüs für die Entwicklungsstufen und das Einbinden mehrerer Leitliniendokumente in eine Liste erläutert. Des Weiteren wurde erklärt, wie die Navigation erstellt wird und die Benutzerverwaltung für dieses Projekt aufgebaut ist. Als Abschluss des Kapitels folgte eine kritische Bewertung der Implementierung dieses Portals mit Ercatons. Hierbei hat sich gezeigt, dass Ercatons für Großprojekte besonders gut geeignet sind, da nachträgliche Änderungen und Erweiterungen des Projektes einfach, schnell und kostengünstig möglich sind. Jedoch hat sich auch herausgestellt, dass momentan kaum Dokumentation vorhanden ist und der Einarbeitungsaufwand in die, zur traditionellen Software-Entwicklung sehr unterschiedliche, Denkweise sehr hoch ist.

8.2 Zukünftige Aufgaben und offene Punkte

Zu den zukünftigen Aufgaben zählt unter anderem eine Abschätzung des Aufwands für die Veränderung der Anzeige von Ercatons. Außerdem ist eine genaue Untersuchung des internen Persistenzmechanismus sinnvoll, da er eine wichtige Eigenschaft der Ercatons darstellt, jedoch in dieser Studienarbeit nur grob umrissen wurde. Ercatons sind laut [ILvW05] transaktionssicher. Deshalb sollte das transaktionale Modell der Ercatons, auf das in dieser Studienarbeit nicht näher eingegangen wurde, ebenfalls untersucht werden. Eine wichtige Aufgabe ist außerdem die Evaluation der Laufzeit-Adaption. Ercatons

können sowohl über das Webportal, als auch über die ESH per „get“- und „put“-Befehl verändert werden. Attributwerte werden zum Beispiel über die Portalsfunktionalität, die Struktur eines Ercatons per „get“ und „put“ verändert. Interessant ist hierbei die Synchronisation der beiden Methoden. Es stellt sich zudem die Frage, wie sich eine Änderung des Java-Quellcodes auf Ercatons, welche auf diesen Code zugreifen, auswirkt. Außerdem gilt es zu klären welche Folgen es hat, wenn ein Benutzer eine Kopie eines Ercatons über die ESH per „put“-Befehl aus der Engine holt, verändert und mit Hilfe von „get“ zurück in die Engine schreibt und dieses Ercaton währenddessen von einem anderen Benutzer bereits verändert wurde.

8.3 Resume

Mit dem Webportal für Patientenleitfäden wurde nur ein kleines Projekt mit Ercatons implementiert, wobei hier bereits die Vorteile gegenüber der klassischen Software-Entwicklung deutlich wurden. Nachträgliche Änderungen an einzelnen Ercatons, ebenso wie an einer Gruppe von Ercatons, welche von einem bestimmten Prototypen abhängen, konnten sehr einfach und sauber realisiert werden. Es war nicht nötig das gesamte Projekt im Voraus zu modellieren, es konnte Schritt für Schritt aufgebaut werden. Die Nachteile der Entwicklung dieses Projektes mit Ercatons, wie die große Anzahl an Klicks, die im Vergleich zur Originalstruktur in Abbildung 7.1 nötig ist, oder der nicht abschätzbare Aufwand zur Veränderung der Anzeige, sind jedoch nicht zu vernachlässigen. Das Verhältnis zwischen Nutzen und dem Einarbeitungsaufwand in eine derart andere Denkweise als bei der bisherigen, traditionelle Software-Entwicklung, ist zudem noch nicht angemessen. Zudem fehlen Dokumentationen, was das Erlernen des Umgangs mit Ercatons deutlich erschwert. Soll jedoch ein Großprojekt entwickelt werden, welches viel Wachstum aufweist, so lohnt es sich dies mit Ercatons zu realisieren. Das problemlose Verändern einzelner Entitäten und der Persistenzmechanismus machen die Ercatons zu einem sehr mächtigen Werkzeug.

A Dokumentation

Im Folgenden werden die Eltern-Tags des Kerns aufgelistet und ein Überblick über die XReferenz, sowie die Java Klasse `com.ercato.lib.util.client.Install` gegeben.

A.1 Erc

Es existieren die folgenden Tags des Kerns:

- id
- type
- role
- base
- clone
- index
- object
- action
- target
- id-ref
- trigger
- version
- catalog
- versions
- permission
- public-for
- version-clone
- prototype-permission

A.2 XReferenz

Die XReferenz ist strukturell wie in A.1 aufgebaut. Hierbei gelten die folgenden Symbolbezeichnungen:
Das `.=` Symbol kennzeichnet Defintionen, `[]` bedeutet „optional“, `<>` stellt einen Platzhalter dar und `{}` eine beliebige `(0..n)` Anzahl von Wiederholungen. Informeller Definitionstext ist in `<< >>` eingeschlossen. Das Pipesymbol | trennt Alternativen und Kommentare beginnen mit `--`.

```
<XReference> := <global_XReference> | <local_XReference>

<global_XReference> := [:<engine>:]<id>[,<version>]{<qualifier>[(<
    parameterlist>)]}[:[<target>][:[<erclet>]]]

<id> := << any legal ercaton-id >>
<version> := v | << any legal version string >>
<target> := null | << any target name >>
<erclet> := null | << any erclet name >>
<qualifier> := ! [(<qtype>)][<qexpression>]
<qtype> := action | target | trigger | xp | xq | scope -- a missing
    qtype defaults to qtype 'action'.
<qexpression> := main | << any qtype expression, e.g., any action name
     >> -- a missing qexpression for qtype 'action' defaults to qname '
    main'
<parameterlist> := <parametername>=<parametervalue> [,<parameterlist>]
<parametername> := << quoted or unquoted name of a parameter >>
<parametervalue> := [(]<XReference>[)] | $[(]<variable>[)] | << quoted
     or unquoted value >>
<engine> := local | << hostname >>[,<engine_id>] -- a missing engine
    defaults to engine 'local' which is local to the current ercaton.
<engine_id> := << port offset with respect to default ports >> -- a
    missing engine_id defaults to engine_id '0', e.g., web port 80.

<local_XReference> := [<path>[,<version>]]{<qualifier>[(<parameterlist
    >)]}[:[<target>][:[<erclet>]]]
```

```
19 <path> := self | this | << id path relative to local ercaton >> -- e.g
     ., '../mysiblingercaton'; a missing path defaults to the local
     ercaton id (self).
```

Quellcode A.1: XReferenz

Die Implementierungen für `<qualifier>` und `<parameterlist>` sind noch nicht in der Engine verwirklicht. Der Platzhalter `<engine>` ist der Zugriff auf Ercatons anderer Engines und ebenfalls noch in Bearbeitung.
Der qtype `xp` (für *XPath*) erlaubt Zugriff auf Teile eines Ercatons. Zum Beispiel würde `~recipe/sample!(xp)/person/name` den Inhalt eines XML-Elementes zurückliefern. Die Option `xq` steht für *XQuery* und `scope` für eine Zugriffsvariante, die enginegestützt robust gegen Refaktorisierungen gehalten wird, z.B. `~recipe/sample!(scope)name`.

A.3 com.ercato.lib.util.client.Install

Diese Klasse installiert und deinstalliert Ercatons in/von einer Ercato Engine. Sie wird, wie in A.2 beschrieben, aufgerufen. Hierbei muss in einer Windows Umgebung anstelle des ':' ein ';' verwendet werden. In `erc.pckgs/libs/erxclient/pdoc/com.ercato.lib.util.client.Install.html` werden die verschiedenen Optionen zum Aufruf der Klasse gezeigt und beschrieben.

```
1 l="<development directory>/libs"
2 java -cp $l/erxclient/erxclient.jar:$l/ErcatoCoreApi-1/esh.jar:$l/
     LivisUtils-1/livisutils1.jar:$l/_Other/woodstox/woodstox.jar:$l/
     _Other/stax/stax.jar com.ercato.lib.util.client.Install [<option>
     ...]
```

Quellcode A.2: com.ercato.lib.util.client.Install - Aufruf

B Webportal Code

In diesem Kapitel wird der Quellcode der Ercatons auszugsweise aufgelistet. Der komplette Quellcode befindet sich auf der beiliegenden CD.

B.1 Webportal-Grundgerüst

```
<?xml version="1.0" encoding="utf-8"?>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:erc="http://ercato.
   com/xmlns/ErcatoCore" xmlns:erx="http://ercato.com/xmlns/
   ErcatoExtensions">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF8"/>
<title>Leitlinien-Webportal</title>
<style type="text/css">
frameset { background: "#0000A0"; }
.sk_fnav  { }
.sk_flist { }
.sk_fwork { }
</style>
   <erx:header xmlns:erx="http://ercato.com/xmlns/ErcatoExtensions"
      xmlns="" xmlns:erc="http://ercato.com/xmlns/ErcatoCore">
      <erc:id>~webportal/start</erc:id>
      <erc:permission role="~any">rxb</erc:permission>
      <erc:target depends-on="setstyle"><erc:arg name="useskin">false<
         /erc:arg>/builtin/default.xsl</erc:target>
      <erc:target name="setstyle"><erc:arg name="style">~webportal/etc
         /style-ruby</erc:arg>/bin/style!setdefaultstyle</erc:target>
   </erx:header>
</head>
<frameset frameborder="no" framespacing="0" border="0" rows="62,*,29">
  <frame src="layout/top.xhtml" marginheight="0" marginwidth="0"/>
```

```
21    <frameset cols="25%,*" frameborder="yes" framespacing="10px" border=
          "10px" bordercolor="#9da1a4">
22    <frameset rows="0,*" frameborder="no" framespacing="0" border="0">
23      <frame src="about:blank" marginheight="0" marginwidth="0"/>
24      <frame src="layout/nav.xhtml" marginheight="0" marginwidth="0"
            scrolling="no" class="sk_fnav" name="nav"/>
25    </frameset>
26    <frameset rows="50%,*" frameborder="yes" framespacing="10px" border=
          "10px" bordercolor="#9da1a4">
27    <frameset rows="0,*" frameborder="no" framespacing="0" border="0">
28      <frame src="about:blank" marginheight="0" marginwidth="0"/>
29      <frame src="home" marginheight="0" marginwidth="0" class="sk_flist
            " name="list"/>
30    </frameset>
31    <frameset rows="*,0" frameborder="no" framespacing="0" border="0">
32      <frame src="layout/empty.xhtml" marginheight="0" marginwidth="0"
            class="sk_fwork" name="work"/>
33      <frame src="about:blank" marginheight="0" marginwidth="0" />
34    </frameset>
35    </frameset>
36    </frameset>
37    <frame src="layout/bottom.xhtml" marginheight="0" marginwidth="0"/>
38    <noframes>Sorry, but HTML frames are required.</noframes>
39  </frameset>
40  </html>
```

Quellcode B.1: ~webportal/start

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <html xmlns="http://www.w3.org/1999/xhtml" xmlns:erc="http://ercato.
        com/xmlns/ErcatoCore" xmlns:erx="http://ercato.com/xmlns/
        ErcatoExtensions">
3     <head>
4       <erx:header xmlns="">
5         <erc:id>~webportal/home</erc:id>
6         <erc:permission role="~any">rxb</erc:permission>
7         <name erc:index="/idx/queries#name">/</name>
8         <category erc:index="/idx/queries#category">#root#</category>
9         <recursive erc:index="/idx/queries#recursive">false</recursive>
```

```
      <erx:objecttype erc:index="/idx/queries#type">homepage</
         erx:objecttype>
      <erc:action name="main">/bin/cat</erc:action>
    </erx:header>
    <title>Home</title>
  </head>
  <body>
    <h1>Leitlinien</h1>
    <p>Dieses Webportal bietet einen Zugriff auf Patientenleitlinien.<
       /p>
    <p>Kopieren, Einfügen und Suchen nach Leitlinien ist möglich.</p>
  </body>
</html>
```

Quellcode B.2: ~webportal/home

```
<?xml version="1.0" encoding="utf-8"?>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:erc="http://ercato.
   com/xmlns/ErcatoCore" xmlns:erx="http://ercato.com/xmlns/
   ErcatoExtensions">
<head>
  <title>Special View</title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  <meta name="styled" content="true"/>
  <erx:apply-css/>
  <script src="/erc/aton/system/activate.js" type="text/javascript"></
     script>
  <erx:header xmlns="">
    <erc:id>~webportal/layout/nav.xhtml</erc:id>
    <erc:permission role="~any">rxb</erc:permission>
  </erx:header>
</head>
<body class="skin_body" id="sksp_mainbody">
<table border="0" cellspacing="0" cellpadding="0" height="100%">
  <!-- the title row -->
  <tr align="left" valign="top">
    <td colspan="3"><table border="0" cellspacing="0" cellpadding="0">
      <tr align="left" valign="top">
      <td id="sksp_title_l"><erx:apply-shim/></td>
```

```
21      <td width="100%" id="sksp_title_m" class="sksp_title"><h1>
            Katalog</h1></td>
22      <td id="sksp_title_r"><erx:apply-shim/></td>
23      </tr>
24    </table></td>
25  </tr>
26  <!-- the toolbar row -->
27  <tr align="left" valign="top">
28    <td id="sksp_toolbar_l"><erx:apply-shim/></td>
29    <td id="sksp_toolbar_m" class="sksp_toolbar"></td>
30    <td id="sksp_toolbar_r"><erx:apply-shim/></td>
31  </tr>
32  <!-- the paper upper border row -->
33  <tr align="left" valign="top">
34    <td id="sksp_canvas_ul"><erx:apply-shim/></td>
35    <td id="sksp_canvas_um"></td>
36    <td id="sksp_canvas_ur"><erx:apply-shim/></td>
37  </tr>
38  <!-- the paper canvas row -->
39  <tr align="left" valign="top" height="100%">
40    <td id="sksp_canvas_ml"></td>
41    <td width="100%" id="sksp_canvas">
42      <script type="text/javascript">
43        p = new Applet("com.livis.naris.applets.NavigatorApplet", "/
              erc/aton/lib",
44            "formslibs.jar", "100%", "100%","CatalogApplet", "0", "0",
                 "middle");
45        p.addParam('configFileName', '/erc/aton/bin/catalog!
              structure::null?catalog-def=~webportal/system/catalog-def')
              ;
46        p.addParam('style', 'color:#9a0000; background-color:#d3d3d3;
              font-family:dialog,Tahoma,arial,helvetica,sans-serif; font-
              size:9px');
47        p.addParam('target', 'list');
48        p.endApplet();
49      </script>
50    </td>
51    <td id="sksp_canvas_mr"></td>
52  </tr>
```

```
53    <!-- the paper lower border row -->
54    <tr align="left" valign="top">
55      <td id="sksp_canvas_ll"><erx:apply-shim/></td>
56      <td id="sksp_canvas_lm"></td>
57      <td id="sksp_canvas_lr"><erx:apply-shim/></td>
58    </tr>
59  </table>
60  </body>
61  </html>
```

Quellcode B.3: ~webportal/layout/nav.xhtml

B.2 Katalog

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <index-table
3     xmlns:erc="http://ercato.com/xmlns/ErcatoCore"
4     xmlns:erx="http://ercato.com/xmlns/ErcatoExtensions"
5   >
6     <erc:type>index</erc:type>
7     <erc:id>~webportal/catalog</erc:id>
8     <erx:permissions>
9       <erc:permission name="~any">rxb</erc:permission>
10    </erx:permissions>
11    <erc:index>
12      <erc:name erx:display="Kategorie">category</erc:name>
13    </erc:index>
14    <erc:index>
15      <erc:name erx:display="Name">name</erc:name>
16    </erc:index>
17    <erc:index>
18      <erc:name>RefCount</erc:name>
19      <erc:index-type>int</erc:index-type>
20      <erc:adder/>
21    </erc:index>
22  </index-table>
```

Quellcode B.4: ~webportal/catalog

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns:erc="http://ercato.com/xmlns/ErcatoCore">
  <description>Standard Navigation</description>
  <erc:id>~webportal/system/catalog-def</erc:id>
  <erc:permission role="~any">rxb</erc:permission>
  <source>
    <index>~webportal/catalog</index>
    <urlbase>/erc/atos/bin/catalog!list?catalog=~webportal/catalog&amp
       ;category=</urlbase>
    <urlappend/>
    <urlencode>true</urlencode>
    <tag>url</tag>
    <type>category</type>
    <filter>
      <column>id</column>
      <value>/builtin/adder</value>
    </filter>
  </source>
    <source>
    <index>/idx/queries</index>
    <urlbase>/erc/atos/bin/query!main?query=</urlbase>
    <urlappend/>
    <urlencode>true</urlencode>
    <tag>query</tag>
    <type>query</type>
    <column>parameter</column>
    <filter>
      <column>type</column>
      <value>query</value>
    </filter>
  </source>
  <source>
    <index>/idx/queries</index>
    <urlbase>/erc/atos</urlbase>
    <urlappend>!main</urlappend>
    <urlencode>false</urlencode>
    <tag>url</tag>
    <type>homepage</type>
```

```
38      <column>parameter</column>
39      <column>target</column>
40      <filter>
41        <column>type</column>
42        <value>homepage</value>
43      </filter>
44    </source>
45  </catalog>
```

Quellcode B.5: ~webportal/system/catalog-def

B.3 Benutzer und Rollen

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <user xmlns:erc="http://ercato.com/xmlns/ErcatoCore">
3    <erc:id>~klaus</erc:id>
4    <erc:type>user</erc:type>
5    <erc:permission role="~admin">rwb</erc:permission>
6    <person>
7      <first-name>Klaus</first-name>
8      <last-name>Testmann</last-name>
9    </person>
10 </user>
```

Quellcode B.6: ~klaus

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <user xmlns:erc="http://ercato.com/xmlns/ErcatoCore">
3    <erc:id>~susi</erc:id>
4    <erc:type>user</erc:type>
5    <erc:permission role="~admin">rwb</erc:permission>
6    <person>
7      <first-name>Susi</first-name>
8      <last-name>Testfrau</last-name>
9    </person>
10 </user>
```

Quellcode B.7: ~susi

```
<?xml version="1.0" encoding="UTF-8"?>
<role
  xmlns:erc="http://ercato.com/xmlns/ErcatoCore"
  xmlns:erx="http://ercato.com/xmlns/ErcatoExtensions"
>
  <erc:id>~webportal/roles/bearbeiter</erc:id>
  <erc:type>role</erc:type>
  <name>Bearbeiter</name>
  <erc:role>
    <erc:granted-for role="~klaus"/>
  </erc:role>
</role>
```

Quellcode B.8: ~webportal/roles/bearbeiter

```
<?xml version="1.0" encoding="UTF-8"?>
<role
  xmlns:erc="http://ercato.com/xmlns/ErcatoCore"
  xmlns:erx="http://ercato.com/xmlns/ErcatoExtensions"
>
  <erc:id>~webportal/roles/betrachter</erc:id>
  <erc:type>role</erc:type>
  <name>Betrachter</name>
  <erc:role>
    <erc:granted-for role="~susi"/>
    <erc:granted-for role="~klaus"/>
  </erc:role>
</role>
```

Quellcode B.9: ~webportal/roles/betrachter

B.4 Base-Ercatons

```
<?xml version="1.0" encoding="UTF-8"?>
<leitlinienteilgebiet
  xmlns:erc="http://ercato.com/xmlns/ErcatoCore"
  xmlns:erx="http://ercato.com/xmlns/ErcatoExtensions"
>
```

```
<erc:id>~webportal/base/leitlinienteilgebiet</erc:id>
<erc:type>prototype</erc:type>
<erc:prototype-permission role="~any">rxb</erc:prototype-permission>
<permissions>
  <erc:permission name="~webportal/roles/betrachter">rb</
     erc:permission>
  <erc:permission name="~webportal/roles/bearbeiter">rxwb</
     erc:permission>
</permissions>
<erc:catalog category="/Leitlinien/Leitlinienteilgebiet" id-ref="~
   webportal/catalog"/>
<leitlinienteilgebiet>
  <name erx:field-ref="string" erx:field-name="Teilgebiet" erx:clear
     -on-copy="true" erc:index="~webportal/catalog#name"></name>
  <columns erx:list-ref="leitlinien" erx:nodename="column" erx:clear
     -on-copy="true">
    <column erc:inherited-as="vector">
      <leitlinienname erx:field-ref="string"  erx:field-name="Name"
           erx:clear-on-copy="true"/>
      <leitlinienlink>
        <erc:id-ref erx:field-ref="idref"  erx:field-name="Link"
           erx:clear-on-copy="true"/>
      </leitlinienlink>
    </column>
  </columns>
</leitlinienteilgebiet>
<erx:actions>
  <erc:target name="annotate">/system/stdann.xsl</erc:target>
  <erc:target name="std" depends-on="annotate" content-type="text/
     xhtml; charset=UTF-8">/system/std.xsl</erc:target>
  <erc:target name="view" depends-on="std" content-type="text/xhtml;
      charset=UTF-8">/system/applets.xsl</erc:target>
  <erc:action erx:field-name="Bearbeiten" default-target="view"
     erx:group="toolbar" name="edit">/bin/edit</erc:action>
  <erc:action erx:field-name="Löschen" erx:group="toolbar" name="
     remove">/bin/rm!wizard</erc:action>
    <erc:action erx:field-name="Neues Leitlinienteilgebiet anlegen"
       erx:group="toolbar" name="copy">/bin/cp!copyAndEdit<erc:arg
       name="clear">true</erc:arg></erc:action>
```

```
32      <erx:triggers>
33        <erc:trigger event="on-change" name="check">/bin/check</
              erc:trigger>
34      </erx:triggers>
35    </erx:actions>
36    <erx:types>
37      <erx:type erc:name="string" lr="string" type="tr-string"/>
38      <erx:type erc:name="idref" lr="idref" type="tr-string"/>
39    </erx:types>
40    <erx:layouts>
41      <erx:layout erc:name="string" method="string" size="30"/>
42      <erx:layout erc:name="idref" method="string" size="60"/>
43    </erx:layouts>
44    <erx:fields>
45      <erx:field erc:name="idref" tr="idref"/>
46      <erx:field erc:name="string" tr="string"/>
47      <erx:list erc:name="leitlinien" field-name="Leitlinien" edit="true
            " nodename="column">
48        <column>
49          <leitlinienname erx:field-ref="string" erx:field-name="Name"/>
50          <leitlinienlink>
51            <erc:id-ref erx:field-ref="idref" erx:field-name="Link"/>
52          </leitlinienlink>
53        </column>
54      </erx:list>
55    </erx:fields>
56  </leitlinienteilgebiet>
```

Quellcode B.10: ~webportal/base/leitlinienteilgebiet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <bibliografischeLeitlinienInfo
3    xmlns:erc="http://ercato.com/xmlns/ErcatoCore"
4    xmlns:erx="http://ercato.com/xmlns/ErcatoExtensions"
5  >
6    <erc:id>~webportal/base/bibliografischeLeitlinienInfo</erc:id>
7    <erc:type>prototype</erc:type>
8    <erc:prototype-permission role="~any">rxb</erc:prototype-permission>
9    <permissions>
```

```
10      <erc:permission name="~webportal/roles/betrachter">rb</
            erc:permission>
11      <erc:permission name="~webportal/roles/bearbeiter">rxwb</
            erc:permission>
12    </permissions>
13    <erc:catalog category="/Leitlinien/Leitlinieninfo" id-ref="~
        webportal/catalog"/>
14    <bibliografischeLeitlinienInfo>
15        <name erx:field-ref="string" erx:field-name="Leitlinienname"
            erx:clear-on-copy="true" erc:index="~webportal/catalog"/>
16        <awmf-reg-nr erx:field-ref="string" erx:field-name="AWMF-Reg.-Nr
            ." erx:clear-on-copy="true"/>
17        <entwicklungsstufe    erx:field-ref="entwicklungsstufe"
            erx:field-name="Entwicklungsstufe" erx:clear-on-copy="true"/>
18        <datum erx:block-ref="date">
19            <aktueller-stand erx:field-ref="date" erx:field-name="
                aktueller Stand" erx:clear-on-copy="true"/>
20            <gueltig-bis erx:field-ref="date" erx:field-name="gültig bis
                " erx:clear-on-copy="true"/>
21      </datum>
22      <columns erx:list-ref="leitliniendokumente" erx:nodename="column"
            erx:clear-on-copy="true">
23        <column erc:inherited-as="vector">
24          <dokumentname erx:field-ref="string" erx:field-name="Name"
                erx:clear-on-copy="true"/>
25          <dokumentlink>
26              <erc:id-ref erx:field-ref="idref" erx:field-name="Link"
                    erx:clear-on-copy="true"/>
27          </dokumentlink>
28        </column>
29      </columns>
30    </bibliografischeLeitlinienInfo>
31    <erx:actions>
32      <erc:target name="annotate" > /system/stdann.xsl  </erc:target>
33      <erc:target name="std" depends-on="annotate" content-type="text/
            xhtml; charset=UTF-8">/system/std.xsl</erc:target>
34      <erc:target name="view" depends-on="std" content-type="text/xhtml;
            charset=UTF-8">/system/applets.xsl</erc:target>
```

```
35      <erc:action erx:field-name="Bearbeiten" default-target="view"
            erx:group="toolbar" name="edit">/bin/edit</erc:action>
36      <erc:action erx:field-name="Löschen" erx:group="toolbar" name="
            remove">/bin/rm!wizard</erc:action>
37        <erc:action erx:field-name="Neue Leitlinieninfo anlegen"
              erx:group="toolbar" name="copy">/bin/cp!copyAndEdit<erc:arg
              name="clear">true</erc:arg></erc:action>
38      <erx:triggers>
39        <erc:trigger event="on-change" name="check"> /bin/check </
              erc:trigger>
40      </erx:triggers>
41    </erx:actions>
42    <erx:types>
43      <erx:type erc:name="string" lr="string" type="tr-string"/>
44      <erx:type erc:name="idref" lr="idref" type="tr-string"/>
45      <erx:type erc:name="date" lr="date" type="tr-date"/>
46    </erx:types>
47    <erx:layouts>
48      <erx:layout erc:name="string" method="string" size="30"/>
49      <erx:layout erc:name="idref" method="string" size="60"/>
50      <erx:layout erc:name="select" method="select-dropdown" size="20"/>
51      <erx:layout erc:name="date" size="10"/>
52    </erx:layouts>
53    <erx:constraints>
54      <erx:constraint erc:name="entwicklungsstufe" source-type="value-
            list" source="~webportal/values/entwicklungsstufe" />
55    </erx:constraints>
56    <erx:fields>
57      <erx:block erc:name="date" field-name="Datum"/>
58      <erx:field erc:name="idref" tr="idref"/>
59      <erx:field erc:name="string" tr="string"/>
60      <erx:field erc:name="date" tr="date"/>
61      <erx:field erc:name="entwicklungsstufe" field-name="
            Entwicklungsstufe" lr="select" cr="entwicklungsstufe"/>
62      <erx:list erc:name="leitliniendokumente" field-name="
            Leitliniendokumente" edit="true" nodename="column">
63        <column>
64          <dokumentname erx:field-ref="string" erx:field-name="Name"/>
65          <dokumentlink>
```

```
          <erc:id-ref erx:field-ref="idref" erx:field-name="Link"/>
        </dokumentlink>
      </column>
    </erx:list>
  </erx:fields>
</bibliografischeLeitlinienInfo>
```

Quellcode B.11: ~webportal/base/bibliografischeLeitlinienInfo

```
<?xml version="1.0" encoding="UTF-8"?>
<leitliniendokument
  xmlns:erc="http://ercato.com/xmlns/ErcatoCore"
  xmlns:erx="http://ercato.com/xmlns/ErcatoExtensions"
>
  <erc:id>~webportal/base/leitliniendokument</erc:id>
  <erc:type>prototype</erc:type>
  <erc:prototype-permission role="~any">rxb</erc:prototype-permission>
  <permissions>
    <erc:permission name="~webportal/roles/betrachter">rb</
       erc:permission>
    <erc:permission name="~webportal/roles/bearbeiter">rxwb</
       erc:permission>
  </permissions>
  <erc:catalog category="/Leitlinien/Leitliniendokument" id-ref="~
     webportal/catalog"/>
  <leitliniendokument>
    <name erx:field-ref="string" erx:field-name="Dokumentname"
       erx:clear-on-copy="true" erc:index="~webportal/catalog#name"></
       name>
    <dokumentlink erx:field-ref="href" erx:field-name="Dokumentlink"
       erx:clear-on-copy="true"/>
  </leitliniendokument>
  <erx:actions>
    <erc:target name="annotate" >/system/stdann.xsl</erc:target>
    <erc:target name="std" depends-on="annotate" content-type="text/
       xhtml; charset=UTF-8">/system/std.xsl</erc:target>
    <erc:target name="view" depends-on="std" content-type="text/xhtml;
        charset=UTF-8">/system/applets.xsl</erc:target>
    <erc:action erx:field-name="Bearbeiten" default-target="view"
       erx:group="toolbar" name="edit">/bin/edit</erc:action>
```

```
23      <erc:action erx:field-name="Löschen" erx:group="toolbar" name="
            remove">/bin/rm!wizard</erc:action>
24      <erc:action erx:field-name="Neues Leitliniendokument anlegen"
            erx:group="toolbar" name="copy">/bin/cp!copyAndEdit<erc:arg
            name="clear">true</erc:arg></erc:action>
25      <erx:triggers>
26        <erc:trigger event="on-change" name="check">/bin/check</
              erc:trigger>
27      </erx:triggers>
28    </erx:actions>
29    <erx:types>
30      <erx:type erc:name="string" lr="string" type="tr-string"/>
31      <erx:type erc:name="href" lr="href" type="tr-string"/>
32    </erx:types>
33    <erx:layouts>
34      <erx:layout erc:name="string" method="string" size="30"/>
35      <erx:layout erc:name="href" method="string" size="60"/>
36    </erx:layouts>
37    <erx:fields>
38      <erx:field erc:name="href" tr="href"/>
39      <erx:field erc:name="string" tr="string"/>
40    </erx:fields>
41  </leitliniendokument>
```

Quellcode B.12: ~webportal/base/leitliniendokument

B.5 Beispiel-Ercatons

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <leitlinienteilgebiet
3     xmlns:erc="http://ercato.com/xmlns/ErcatoCore"
4     xmlns:erx="http://ercato.com/xmlns/ErcatoExtensions"
5   >
6     <erc:clone>~webportal/base/leitlinienteilgebiet</erc:clone>
7     <erc:id>~webportal/sample/Allgemeine-Gynaekologie</erc:id>
8     <leitlinienteilgebiet>
9       <name>Allgemeine Gynäkologie</name>
10      <columns>
```

```
      <column>
        <leitlinienname>Perioperative Antibiotikaprophylaxe</
           leitlinienname>
            <leitlinienlink>
                <erc:id-ref>~webportal/sample/Allgemeine-Gynaekologie/
                   Perioperative-Antibiotikaprophylaxe</erc:id-ref>
            </leitlinienlink>
        </column>
        <column>
            <leitlinienname>Prophylaxe der venösen Thromboembolie</
               leitlinienname>
            <leitlinienlink>
                <erc:id-ref>~webportal/sample/Allgemeine-Gynaekologie/
                   Prophylaxe-der-venoesen-Thromboembolie</erc:id-ref>
            </leitlinienlink>
        </column>
    </columns>
  </leitlinienteilgebiet>
</leitlinienteilgebiet>
```

Quellcode B.13: ~webportal/sample/Allgemeine-Gynaekologie

```
<?xml version="1.0" encoding="UTF-8"?>
<leitlinienteilgebiet
  xmlns:erc="http://ercato.com/xmlns/ErcatoCore"
  xmlns:erx="http://ercato.com/xmlns/ErcatoExtensions"
>
  <erc:clone>~webportal/base/leitlinienteilgebiet</erc:clone>
  <erc:id>~webportal/sample/Gynaekologische-Onkologie</erc:id>
  <leitlinienteilgebiet>
    <name>Gynäkologische Onkologie</name>
    <columns>
        <column>
            <leitlinienname>Endometriumkarzinom</leitlinienname>
            <leitlinienlink>
                <erc:id-ref>~webportal/sample/Gynaekologische-
                   Onkologie/Endometriumkarzinom</erc:id-ref>
            </leitlinienlink>
        </column>
        <column>
```

```
            <leitlinienname>Zervixkarzinom</leitlinienname>
            <leitlinienlink>
                <erc:id-ref>~webportal/sample/Gynaekologische-
                    Onkologie/Zervixkarzinom</erc:id-ref>
            </leitlinienlink>
        </column>
    </columns>
  </leitlinienteilgebiet>
</leitlinienteilgebiet>
```

Quellcode B.14: ~webportal/sample/Gynaekologische-Onkologie

```
<?xml version="1.0" encoding="UTF-8"?>
<bibliografischeLeitlinienInfo
  xmlns:erc="http://ercato.com/xmlns/ErcatoCore"
  xmlns:erx="http://ercato.com/xmlns/ErcatoExtensions"
>
  <erc:clone>~webportal/base/bibliografischeLeitlinienInfo</erc:clone>
  <erc:id>~webportal/sample/Allgemeine-Gynaekologie/Perioperative-
     Antibiotikaprophylaxe</erc:id>
  <bibliografischeLeitlinienInfo>
    <name>Perioperative Antibiotikaprophylaxe </name>
    <awmf-reg-nr>029/022</awmf-reg-nr>
    <entwicklungsstufe erx:display="Entwicklungsstufe 1">1</
       entwicklungsstufe>
    <datum>
        <aktueller-stand>02/2004</aktueller-stand>
        <gueltig-bis>02/2009</gueltig-bis>
    </datum>
    <kommentar erx:field-name="Kommentar">(Federführung: AK "
       Krankenhaushygiene" der AWMF)</kommentar>
    <columns>
        <column>
            <dokumentname>Leitlinie</dokumentname>
            <dokumentlink>
                <erc:id-ref>~webportal/sample/Allgemeine-Gynaekologie/
                    Perioperative-Antibiotikaprophylaxe/Leitlinie</
                    erc:id-ref>
            </dokumentlink>
        </column>
```

```
    </columns>
  </bibliografischeLeitlinienInfo>
</bibliografischeLeitlinienInfo>
```

Quellcode B.15: ~webportal/sample/Allgemeine-Gynaekologie/Perioperative-Antibiotikaprophylaxe

```
<?xml version="1.0" encoding="UTF-8"?>
<bibliografischeLeitlinienInfo
  xmlns:erc="http://ercato.com/xmlns/ErcatoCore"
  xmlns:erx="http://ercato.com/xmlns/ErcatoExtensions"
>
  <erc:clone>~webportal/base/bibliografischeLeitlinienInfo</erc:clone>
  <erc:id>~webportal/sample/Allgemeine-Gynaekologie/Prophylaxe-der-
     venoesen-Thromboembolie</erc:id>
  <bibliografischeLeitlinienInfo>
    <name>AWMF-Leitlinie Prophylaxe der venösen Thromboembolie (VTE) <
       /name>
    <awmf-reg-nr>003/001</awmf-reg-nr>
    <entwicklungsstufe erx:display="Entwicklungsstufe 3">3</
       entwicklungsstufe>
    <datum>
        <aktueller-stand>04/2009</aktueller-stand>
        <gueltig-bis>12/2013</gueltig-bis>
    </datum>
    <columns>
        <column>
            <dokumentname>Langfassung</dokumentname>
            <dokumentlink>
                <erc:id-ref>~webportal/sample/Allgemeine-Gynaekologie/
                   Prophylaxe-der-venoesen-Thromboembolie/Langfassung<
                   /erc:id-ref>
            </dokumentlink>
        </column>
        <column>
            <dokumentname>Kurzfassung</dokumentname>
            <dokumentlink>
                <erc:id-ref>~webportal/sample/Allgemeine-Gynaekologie/
                   Prophylaxe-der-venoesen-Thromboembolie/Kurzfassung<
                   /erc:id-ref>
```

```
            </dokumentlink>
        </column>
    </columns>
  </bibliografischeLeitlinienInfo>
</bibliografischeLeitlinienInfo>
```

Quellcode B.16: ~webportal/sample/Allgemeine-Gynaekologie/Prophylaxe-der-venoesen-Thromboembolie

```
<?xml version="1.0" encoding="UTF-8"?>
<leitliniendokument
  xmlns:erc="http://ercato.com/xmlns/ErcatoCore"
  xmlns:erx="http://ercato.com/xmlns/ErcatoExtensions"
>
  <erc:clone>~webportal/base/leitliniendokument</erc:clone>
  <erc:id>~webportal/sample/Allgemeine-Gynaekologie/Perioperative-
     Antibiotikaprophylaxe/Leitlinie</erc:id>
  <leitliniendokument>
    <name>Leitlinie</name>
    <dokumentlink>http://www.uni-duesseldorf.de/AWMF/ll/029-022.htm</
       dokumentlink>
  </leitliniendokument>
</leitliniendokument>
```

Quellcode B.17: ~webportal/sample/Allgemeine-Gynaekologie/Perioperative-Antibiotikaprophylaxe/Leitlinie

```
<?xml version="1.0" encoding="UTF-8"?>
<leitliniendokument
  xmlns:erc="http://ercato.com/xmlns/ErcatoCore"
  xmlns:erx="http://ercato.com/xmlns/ErcatoExtensions"
>
  <erc:clone>~webportal/base/leitliniendokument</erc:clone>
  <erc:id>~webportal/sample/Allgemeine-Gynaekologie/Prophylaxe-der-
     venoesen-Thromboembolie/Kurzfassung</erc:id>
  <leitliniendokument>
    <name>Kurzfassung</name>
    <dokumentlink>http://www.uni-duesseldorf.de/AWMF/ll/003-001k.pdf</
       dokumentlink>
  </leitliniendokument>
```

```
</leitliniendokument>
```

Quellcode B.18: ~webportal/sample/Allgemeine-Gynaekologie/Prophylaxe-der-venoesen-Thromboembolie/Kurzfassung

B.6 Suchabfrage

```
<?xml version="1.0" encoding="utf-8"?>
<query
  xmlns:erx="http://ercato.com/xmlns/ErcatoExtensions"
  xmlns:erc="http://ercato.com/xmlns/ErcatoCore"
  erx:check="valid"
>
  <erc:clone>/lib/base/query</erc:clone>
  <erc:id>~webportal/query/leitliniensuche</erc:id>
  <erc:permission name="~any"/>
  <name>/Suchabfragen/Leitliniensuche</name>
  <description>Suche nach Leitlinieninfos</description>
  <query>
    <index>
      <erc:id-ref>~webportal/catalog</erc:id-ref>
    </index>
    <columns>
      <column>
        <name>name</name>
        <sort>asc</sort>
        <width/>
        <idrefcol>id</idrefcol>
      </column>
      <column>
        <name>category</name>
      </column>
    </columns>
    <filters>
      <or>
        <and>
          <column>name</column>
          <firstletter>true</firstletter>
```

```
32          </and>
33        </or>
34      </filters>
35    <autoquery>true</autoquery>
36    <pagesize erx:field-ref="pagesize">200</pagesize>
37    </query>
38    <catalog>
39          <erc:catalog category="/Suchabfragen/"/>
40    </catalog>
41    <erx:actions>
42        <erc:target name="list">/system/values2list.xsl</erc:target>
43     </erx:actions>
44  </query>
```

Quellcode B.19: ~webportal/query/leitliniensuche

Literaturverzeichnis

[Amb03] Scott Ambler. *Agile Database Techniques: Effective Strategies for the Agile Software Developer*. Wiley Publishing, 2003.

[Bor86] A. H. Borning. Classes versus prototypes in object-oriented languages. In *ACM '86: Proceedings of 1986 ACM Fall joint computer conference*, pages 36–40, Los Alamitos, CA, USA, 1986. IEEE Computer Society Press.

[dWMFe09] Arbeitsgemeinschaft der Wissenschaftlichen Medizinischen Fachgesellschaften e.V. Awmf online: http://www.uni-duesseldorf.de/awmf/, 2009.

[ECM09] Ecma-262 edition 5 standard. http://www.ecma-international.org/publications/files/ecma-st/ecma-262.pdf, December 2009.

[Hor01] Waldemar Horwat. Javascript 2.0: Evolving a language for evolving systems. In *Lightweight Languages Workshop (LL1)*, http://www.mozilla.org/js/language/evolvingJS.pdf, 2001.

[IL09] Oliver Imbusch and Falk Langhammer. Ercatons and organic programming, 2009.

[ILvW04] Oliver Imbusch, Falk Langhammer, and Guido von Walter. Ercatons: Thing-oriented programming. In *Lecture Notes in Computer Science LNCS 3263, Springer Verlag*, 2004.

[ILvW05] Oliver Imbusch, Falk Langhammer, and Guido von Walter. Ercatons and organic programming: say good-bye to planned economy. In *OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 41–52, New York, NY, USA, 2005. ACM.

[Liv09a] Ercatoj engine. http://www.living-pages.de/de/products/ercatoj/, 2009.

[Liv09b] Xoperator evaluation kit. http://www.living-pages.de/de/projects/xop/, 2009.

[LR06] Bernhard Lahres and Gregor Rayman. *Praxisbuch Objektorientierung, Von den Grundlagen zur Umsetzung*. Galileo Computing, 2006.

[MT07] Tommi Mikkonen and Antero Taivalsaari. Using javascript as a real programming language. 2007.

[SU95] Randall B. Smith and David Ungar. Programming as an experience: The inspiration for self. In *ECOOP '95 Conference Proceedings, Aarhus, Denmark, August, 1995.*, 1995.

[US87] David Ungar and Randall B. Smith. Self: The power of simplicity. In *OOPSLA '87 Conference Proceedings, pp. 227-241, Orlando, FL, October, 1987*, 1987.