Studienarbeit-Nr. 2022

Konzeption einer internetbasierten Datenbankanwendung zur Verwaltung und Analyse von Patienten- und Krankheitsdaten

Andrea Pachner

Studiengang:	Informatik
Prüfer/in:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer/in:	Fabian Kaiser
Ext. Betreuer:	Friedhelm Brinkmann (OSP)
Beginn am:	01.06.2005
Beendet am:	30.11.2005
CR-Nummer:	H.5.2, H.5.3 und J.3.

Inhaltsverzeichnis

1. Aufgabenstellung 4
2. Arbeitsumfeld 5
2.1. Der Onkologische Schwerpunkt Stuttgart (OSP) 5
2.1.1. Interdisziplinäre Kooperationsforen 6
2.1.2. Krebsgemeinschaft.de 6
2.1.3. Bückenschwestern® 7
2.1.4. Brustzentren 7
2.1.5. Qualitäts- und Datenmanagement 7
2.2. Tumordokumentation 10
3. Bisheriges System und Caché 13
3.1. MUMPS 14
3.2. KRAZTUR 15
3.3. Caché 18
4. Überführung der Daten 25
4.1. Die Portierung der Daten von MUMPS nach Caché 25
4.2. Probleme bei der Überführung der Daten 27
5. Möglichkeiten der Umsetzung 29
5.1. Client/Server-Applikation 30
5.1.1. Analyse 30
5.1.2. Evaluation 32
5.2. Web Services 33
5.2.1. Analyse 33
5.2.2. Evaluation 35
5.3. Web Anwendung mittels Caché Server Pages 37
5.3.1. Analyse und Vorstellung 37
5.3.2. Evaluation 42
5.4. Fazit 44
6. Durchführung 47
6.1. Vorgehensweise 47
6.1.1. Anforderungsanalyse 47
6.1.2. Datenportierung 49
6.1.3. Oberfläche 49
6.2. Erste Komponenten 52
6.2.1. Kurzersterfassung in KRAZTUR 52
6.2.2. Die neue graphische Oberfläche 54
7. Zusammenfassung 57
8. Literatur 59
9. Anhang 60
9.1. Klassendefinition der Kurzersterfassung 60
9.1.1. Klasse KERST 60
9.1.2. Klasse KEbeharzt 62
9.1.3. Klasse KEhisto 63
9.1.4. Klasse KElok 64
9.1.5. Klasse KEmeta 64
9.1.6. Klasse KEaend 65
Versicherung 66

Abstract

Der Onkologische Schwerpunkt Stuttgart e.V. (OSP) ist ein gemeinnütziger Verein, der sich zur Aufgabe gemacht hat, die Situation Krebskranker in der Region Stuttgart zu optimieren. Seit 1988 erfasst und speichert der OSP die Daten zu Diagnose, Therapie, Krankheitsverlauf und Nachsorge aller Tumorpatienten mittels eines zentralen Tumordokumentations- und Informationssystems (KRAZTUR). Aus den gesammelten Daten sollen wissenschaftliche Erkenntnisse für einheitliche Standards und medizinische Innovationen gewonnen werden. Ein leistungsfähiges Datenmanagement ist deshalb von größter Bedeutung. Bisher wurde als Datenbank und Programmiersprache MUMPS eingesetzt, eine in der Medizininformatik bewährte Datenverarbeitungstechnologie. 2004 erwarb der OSP die Lizenz für eine Caché-Datenbank. Die Portierung der Daten musste von MUMPS nach Caché erfolgen, was nicht ohne Probleme gelang (Kapitel 4). An dieser Stelle war auch zu prüfen, welche Teile der alten Datenbank beibehalten werden konnten.

Aufgabe dieser Studienarbeit ist es, die beste Möglichkeit herauszuarbeiten, um die bisher zeichenorientierte Anwendung durch eine graphische, benutzerfreundliche Applikation zu ersetzen. Nach Erörterung der Vor- und Nachteile der zweischichtigen Client/Server-Lösung, der Web Services und der Caché Server Pages (CSP) ist die von InterSystems entwickelte CSP-Technologie den anderen vorzuziehen, da bei dieser Umsetzungsmöglichkeit nichts auf dem Client installiert werden muss.

Am Ende dieser Studienarbeit wird eine erste Komponente der graphischen Oberfläche vorgestellt.

1. Aufgabenstellung

Aufgabe dieser Studienarbeit ist es, für die Tumordatenbank am Onkologischen Schwerpunkt Stuttgart (OSP) eine benutzerfreundlichere Oberfläche zu erstellen. Die jetzige zeichenorientierte Dialoganwendung soll auf eine moderne internetbasierte Anwendung umgestellt werden. Dafür stehen drei Realisierungsalternativen zur Verfügung, die analysiert und evaluiert werden müssen. Die neue Oberfläche könnte als Client/Server-Applikation, auf Basis von Web Services oder als Web-Anwendung mittels Caché Server Pages realisiert werden.
Möglich wurde die Umstellung der Oberfläche am OSP durch die Ablösung der alten Datenbank MUMPS durch Caché im Jahr 2004.
Hierzu wird als weitere Aufgabe der Studienarbeit die Portierung der Daten von der Datenbank M nach Caché und die dabei auftretenden Fehler beschrieben.
Als Grundlage für die Umsetzung dieser Arbeit durch den OSP dienen erste Komponenten, die im Rahmen dieser Studienarbeit entwickelt wurden.

2. Arbeitsumfeld

2.1. Der Onkologische Schwerpunkt Stuttgart (OSP)

Im Onkologischen Schwerpunkt Stuttgart e.V. [OSP1] [OSP2], der seinen Sitz im Diakonieklinikum hat, haben sich 14 Stuttgarter Kliniken zu einem gemeinnützigen Verein zusammengeschlossen. Die Aufgabe des OSP ist es, die Situation Krebskranker in der Region Stuttgart zu optimieren. Um dies zu erreichen, wird die Zusammenarbeit zwischen den Kliniken in Stuttgart und der Region sowie den niedergelassenen Ärzten intensiviert und Fortbildungen werden angeboten. Zudem wurden für Ärzte und Pflegende interdisziplinäre Kommunikations- und Kooperationsforen eingerichtet, um gemeinsam Strategien und Innovationen zu entwickeln. Die Brückenschwestern sind einbezogen, um die häusliche Betreuung schwerkranker Tumorpatienten zu übernehmen. Zur Qualitätssicherung werden Fallbesprechungen durchgeführt und Therapieleitlinien erstellt. Gemessen und ausgewertet werden kann die Ergebnisqualität durch ein EDV-gestütztes Tumordokumentations- und Informationssystem.

Im Organigramm (Abb. 2.1.) ist die Organisationsstruktur des OSP dargestellt.

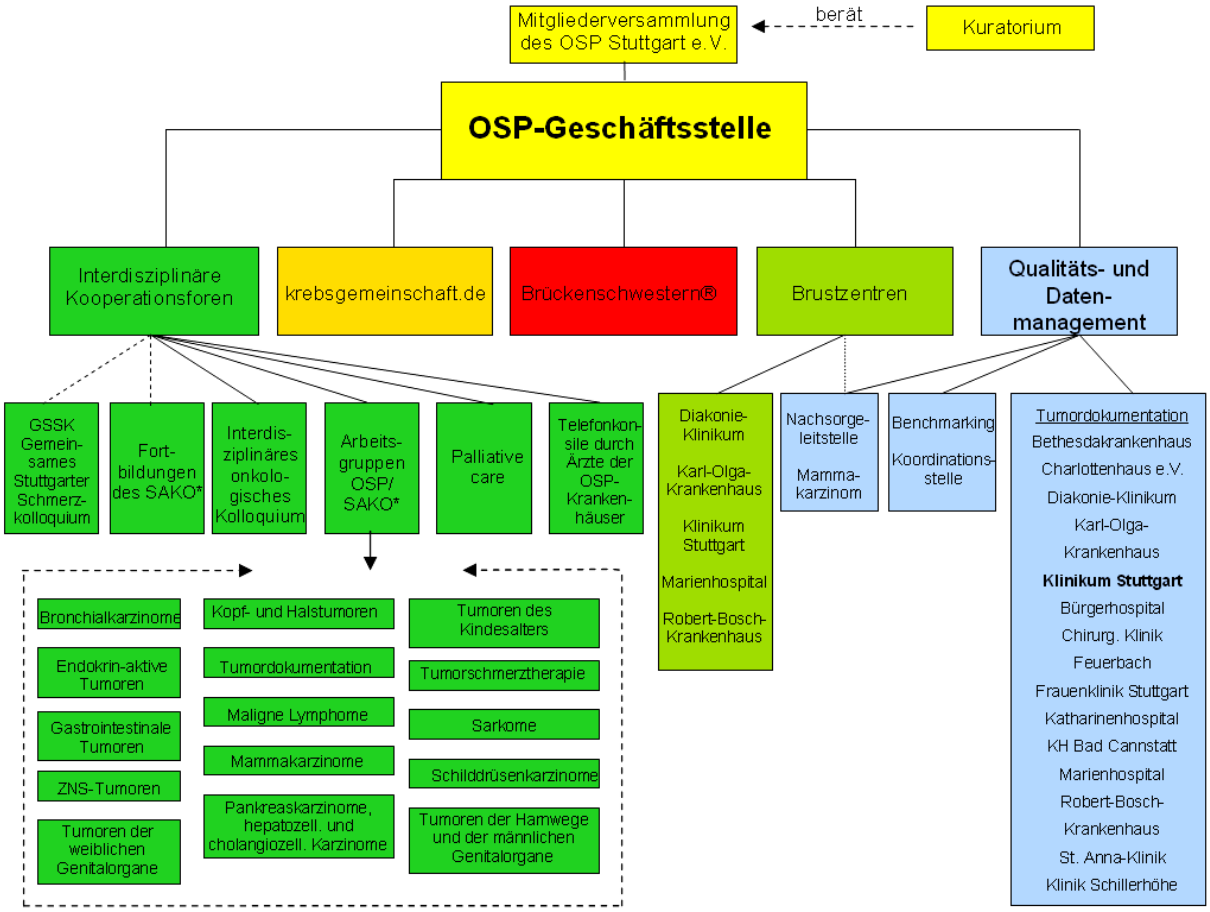

Abb. 2.1. Organisationsstruktur des Onkologischen Schwerpunkt Stuttgarts [OSP2]
*SAKO = Stuttgarter Arbeitskreis für klinische Onkologie e.V.

2.1.1. Interdisziplinäre Kooperationsforen

Interdisziplinäres onkologisches Kolloquium

Im interdisziplinären onkologischen Kolloquium treffen sich Spezialisten verschiedener Fachrichtungen, um die bestmögliche Behandlung für Patienten mit ungewöhnlichen Krankheitsbildern zu besprechen. Für Ärzte ist das Kolloquium gleichzeitig eine Fortbildung und dient der Qualitätssicherung.

Krankheitsbezogene Arbeitsgruppen

In den krankheitsbezogenen Arbeitsgruppen werden Leitlinien für Diagnostik und Therapie verschiedener Tumorentitäten aktualisiert und erarbeitet. Diese Leitlinien beschreiben den Versorgungsstandard in den Stuttgarter OSP-Krankenhäusern. Außerdem werden klinikübergreifende anonymisierte Auswertungen der Daten aus der Tumor-Datenbank in Auftrag gegeben.

Palliative care

Unter Palliative care versteht man die Behandlung und Begleitung von Patienten mit einer unheilbaren, weit fortgeschrittenen Erkrankung und begrenzter Lebenserwartung. Die Palliativmedizin lehnt aktive Sterbehilfe ab. Stattdessen versucht sie durch eine ganzheitliche Behandlung das Leiden der Patienten zu lindern, ihnen und ihren Angehörigen bei der Bewältigung der Krankheit zu helfen und die Lebensqualität zu verbessern. Dazu kooperieren Ärzte verschiedener Fachrichtungen, das Krankenhauspersonal und andere Berufsgruppen, die Menschen mit unheilbaren Krankheiten stationär oder ambulant betreuen.

Telefonkonsile durch Ärzte der OSP-Krankenhäuser

Dieses Angebot richtet sich an Krankenhausärzte und niedergelassene Ärzte. Es bietet einen kontinuierlichen, telefonischen Kontakt zu Fachärzten aller onkologischen Fachrichtungen an.

2.1.2. Krebsgemeinschaft.de

Im Rahmen des Projektes COSMOS (Community Online Sevices and Mobile Solutions) entstanden, wendet sich der Internetauftritt www.krebsgemeinschaft.de an Frauen mit Brustkrebs und alle an diesem Thema Interessierte. Es stellt qualitätsgesicherte Informationen kostenlos und werbefrei zur Verfügung und bietet den Interessierten untereinander und mit Fachärzten Austauschmöglichkeiten. Zudem wird auf Veranstaltungen und Literatur zum Thema Brustkrebs hingewiesen. Es besteht auch die Möglichkeit zum Chatten und Erfahrungsberichte ins Netz zu stellen.

2.1.3. Bückenschwestern®

Die Brückenschwestern betreuen und begleiten schwerkranke Tumorpatienten, die ihre letzte Lebenszeit im häuslichen Umfeld verbringen möchten, und deren Angehörige. Sie können bei medizinisch-pflegerischen und psychosozialen Problemen beraten und durch eine gute palliative Pflege Krankenhausaufenthalte vermeiden oder wenigstens verkürzen.

2.1.4. Brustzentren

Die fünf Brustzentren der Stuttgarter OSP-Krankenhäuser haben sich 2003 zusammengeschlossen, um eine optimale und qualitätsgesicherte Behandlung von Mammakarzinompatientinnen (Frauen mit Brustkrebs) zu gewährleisten. Durch gemeinsame Leitlinien in der Behandlung, gemeinsame Studien und Fortbildungen soll die Behandlungsqualität bei Brustkrebs gesteigert werden. Der Aufbau von Doppelstrukturen soll durch den Zusammenschluss vermieden werden, vielmehr sollen vorhandene Strukturen gemeinsam genutzt und optimiert werden. Die Kooperation mit niedergelassenen Ärzten und Radiologen soll zur Optimierung der Primärdiagnostik und Nachsorge und damit zu einer qualitätsgesicherten ganzheitlichen Patientenversorgung beitragen.

2.1.5. Qualitäts- und Datenmanagement

Die Basis des Qualitäts- und Datenmanagements bildet die Tumordokumentation. Da sie der Gegenstand dieser Studienarbeit ist, wird sie gesondert in Kapitel 2.2. vorgestellt. Auf diesem klinischen Krebsregister baut die Arbeit des Benchmarking im OSP auf. Auch die Nachsorgeleitstelle benutzt die Daten, um im Rahmen der Mammakarzinom-Nachsorge-Studie die Ergebnisqualität der Behandlung an den OSP-Krankenhäusern zu untersuchen.

Nachsorgeleitstelle

Der häufigste bösartige Tumor in der weiblichen Bevölkerung hierzulande ist das Mammakarzinom (Abb. 2.2.). Trotz der erfolgreichen operativ-kurativen Behandlung, kommt es bei circa 60% der Patientinnen innerhalb von 10 Jahren zum Rezidiv – einem Rückfall. Wenn das Rezidiv lokoregionär ist und keine Fernmetastasen auftreten, sind die Heilungschancen umso besser, je früher das Rezidiv erkannt wird. Daher ist die Nachsorge so wichtig. Um die Patientinnen bei der Nachsorge zu begleiten, ist die Nachsorgeleitstelle in Zusammenarbeit mit dem Institut für Medizinische Informationsverarbeitung der Universität Tübingen (IMI) im Rahmen der Feldstudie „Qualitätssicherung in der Mammakarzinom-Nachsorge“ entstanden. Diese Feldstudie hatte zum Ziel, die konventionelle Nachsorgeform mit der symptomorientierten zu vergleichen und ein Qualitätsmanagement für die Mammakarzinom-Nachsorge aufzubauen. Durch diese Feldstudie konnte belegt werden, dass die

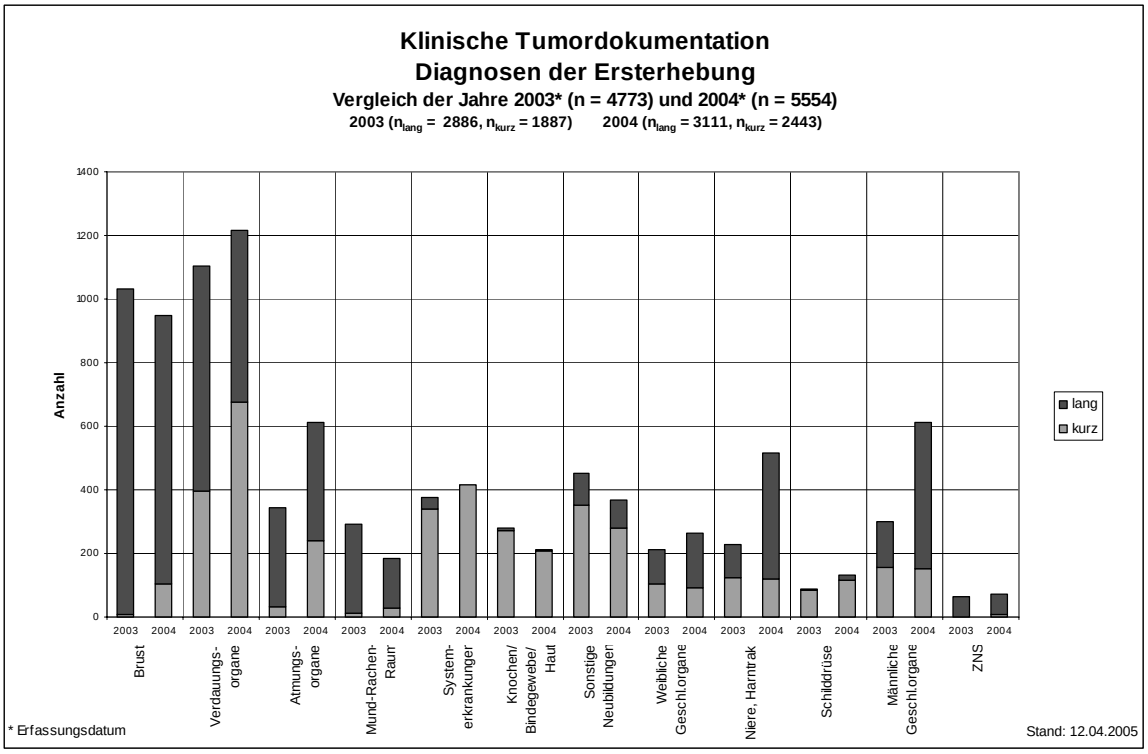

Abb. 2.2. klinische Tumordokumentation, Diagnosen der Ersterhebung aus dem Jahresbericht 2004 [OSP2] aufgeteilt in Lang- und Kurzdokumentation (siehe Kapitel 2.2.)

konventionelle, apparative Nachsorge für die Patientinnen in Bezug auf Gesamtüberlebenszeit, Lebensqualität und Entdeckung von Rezidiven keine Vorteile bringt. Aufgrund dessen und des Kostenvergleichs der beiden Nachsorgearten wird diese Nachsorgeart nicht mehr angeboten.

Da die wissenschaftlichen Erkenntnisse über die Behandlung und die Nachsorge von Brustkrebs – wie nach Auswertung dieser Feldstudie – ständig zunehmen, sind die Information darüber und die Kommunikation für Patientinnen und Ärzte essentiell. Nur so kann der international anerkannte Standard in der Nachsorge durch einen reibungslosen Informationsfluss zwischen Kliniken und niedergelassenen Ärzten gewährleistet werden. Die Brustkrebspatientinnen in der Region Stuttgart werden dabei von

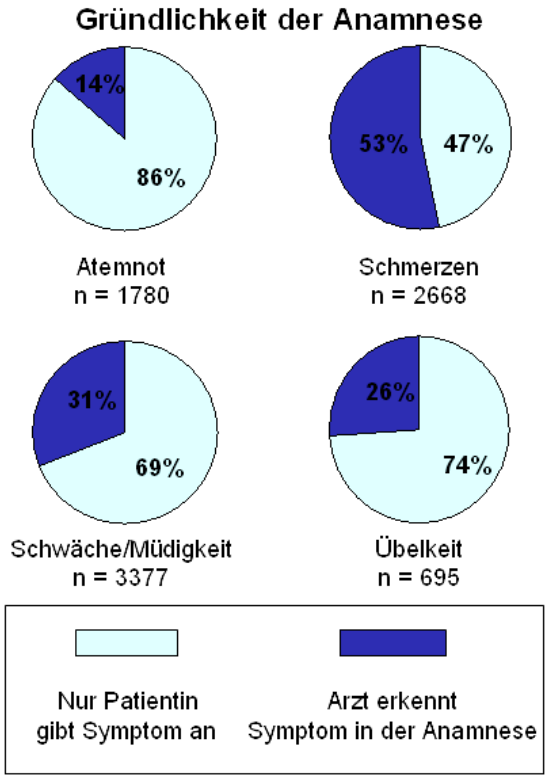

Abb.2.3. Gründlichkeit der Anamnese aus Jahresbericht 2001

der Nachsorgeleitstelle durch die qualitätsgesicherte Nachsorge begleitet und in dieser Zeit betreut. Auch eine Beratung steht ihnen zur Verfügung.
Um die Qualität in der Behandlung zu sichern, bekommen die primär behandelnden Kliniken einen Datenrücklauf und können so ihre Arbeit rückblickend beurteilen. Die nachsorgenden Ärzte erhalten von der Nachsorgeleitstelle Checklisten und Erinnerungsschreiben, die Patientinnen – wenn erwünscht – Informationen über Veranstaltungen, Rundschreiben und persönliche Terminerinnerungen. Ein weiterer Teil der Arbeit der Nachsorgeleitstelle ist die Dokumentation des Lebensqualitätsfragebogens, der bei jeder Nachsorge von den Patientinnen ausgefüllt wird. Bei der Auswertung zeigt sich, wie viel Information bei der meist unter Zeitdruck durchgeführten Erhebung der Krankengeschichte - der Anamnese - verloren gehen kann (Abb. 2.3.). Dieser Fragebogen ergänzt daher sinnvoll die Anamnese durch den Arzt. Erst durch die lückenlose Dokumentation ist das Erkennen dieser Diskrepanz und anderer Fehler möglich geworden. Nur mit ihrer Hilfe können die Gründe analysiert werden, warum manche anerkannte Leitlinien nicht angewendet werden, und es können die notwendigen Veränderungen eingeleitet werden, um die Qualität in der Nachsorge sicherzustellen.

Benchmarking

Das neueste Projekt am OSP ist das Benchmarking: Leistungsvergleich und Lernen von den Besten. Es wurde möglich durch die konsequente Umstellung der klinischen Tumordokumentation auf die Ergebnisqualitätssicherung im Jahr 2002. Nachdem die gesetzlichen Grundlagen dafür geschaffen wurden, konnte 2003 der Vitalstatus der dokumentierten Patienten zum ersten Mal mit den Melderegistern der Regionalen Rechenzentren des Landes abgeglichen werden. Über die zurückgemeldeten Parameter *lebend, verzogen, verstorben und nicht auffindbar* können Aussagen über die Ergebnisqualität in Hinblick auf die Überlebenszeit erstmalig für alle dokumentierten Patienten gemacht werden. Neben diesem wichtigen Parameter für die Messung der Ergebnisqualität ist die rezidivfreie, die rückfallsfreie Zeit ein weiterer Faktor, der mittels eines Datenrücklaufsystems ermittelt wird. Regelmäßige Auswertungen und Analysen dieser Daten können Verbesserungsmöglichkeiten aufzeigen und der Anstoß für geeignete Veränderungsmaßnahmen sein. Über die von den niedergelassenen Ärzten ausgefüllten Datenrücklaufbögen, die einmal jährlich Verlaufinformationen zu den Patientinnen und Patienten liefern, sollen die niedergelassenen Ärzte in die Tumordokumentation mit eingebunden werden, um die Versorgung der Tumorpatienten langfristig zu optimieren.
Das Datenrücklaufsystem ist Kern des Benchmarking-Projekts. Es hat sich zum Ziel gesetzt, eine konsequente Messung der Ergebnisqualität durchzuführen, eine nachhaltige Verbesserung der Patientenversorgung zu erreichen, die Daten national und international zu vergleichen und das Benchmarking in OSP-Krankenhäusern und der Ärzteschaft zu etablieren. Dafür werden exemplarisch die Daten von Mammakarzinom, Kolonkarzinom und Rektumkarzinom im Hinblick auf Überlebenszeit und rezidivfreie Zeit in Relation zur

Tumorklassifikation nach TNM (Stadieneinteilung bezüglich der Ausdehnung des Primärtumors [T] dem Befall der regionalen Lymphknoten = Nodus [N] und der Existenz von Fernmetastasen [M]) in Zusammenhang mit der Lebensqualität erhoben und vom IMI anonymisiert ausgewertet. Anhand dieser Auswertungen können Probleme erfasst und durch „Lernen von den Besten" Lösungswege entwickelt werden.
Für die Erhebung der Daten ist das Einverständnis des Patienten unerlässlich. Dazu bekommt er zusammen mit Lebensqualitätsbögen, die jährlich ausgefüllt werden sollen, eine Einverständniserklärung.
Nur durch die Zusammenarbeit von Patienten und Patientinnen, Ärzten, den OSP-Krankenhäusern und dem OSP können die Krankheitsverläufe der Patienten anhand der Daten verfolgt und aussagekräftige Statistiken erhoben werden, die für den Vergleich der Behandlungen nötig sind.

2.2. Tumordokumentation

Seit 1988 werden die Daten zu Diagnostik, Therapie, Krankheitsverlauf und Nachsorge aller Tumorpatienten in der Region Stuttgart, die in einem OSP-Krankenhaus behandelt werden, mittels eines zentralen Tumordokumentations- und Informationssystems (KRAZTUR) erfasst und gespeichert. Die Speicherung und Verarbeitung der OSP-Daten findet getrennt in den klinikeigenen Datenbeständen statt. Zu qualitätssichernden, allgemeinen Auswertungen werden die Daten anonymisiert zusammengeführt. Da die zusammenfassende Verlaufsdarstellung wenig Resultate für die regelmäßige Ergebnisqualität lieferte, wurde die EDV-gestützte klinische Tumordokumentation 2002 konsequent auf Ergebnisqualitätssicherung umgestellt. Es fehlten nämlich Angaben zur Nachsorge und zum Vitalstatus der Patienten, die jetzt mit standardisierten Fragebögen ermittelt werden. Dazu wurde - wie schon beschrieben - ein Datenrücklaufsystem etabliert und nach Schaffung der rechtlichen Grundlagen der Melderegisterabgleich eingeführt. Mangels personeller Ressourcen können nicht alle Tumorentitäten gleichermaßen dokumentiert werden, so werden manche Tumorarten nur in Kurzform, in der Kurzerhebung (Abb. 2.4.) dokumentiert.
Die Kurzersterhebung wird im Kapitel 6 in die neue graphische Oberfläche umgesetzt.

	Kurzdokumentation	Langdokumentation
Patientenstammdaten (Name, Vorname, Geburtsname, Geburtsdatum, Geschlecht, Adresse)	X	X
Krankenkasse	X	X
Studienteilnahme	(X)	(X)
Haus- und weitere Ärzte	--	X
WHO (Karnofsky) (Allgemeinzustand)	--	X
1. ärztliche Diagnose	--	(X)
ICD (International Classification of Diseases) (Diagnoseklassifikation)	X	X
ICD, Bemerkung	(X)	(X)
Diagnosedatum	X	X
Menopausen-Status	--	X[M]
Klinischer TNM (Tumorklassifikation)	--	X
Pathologischer TNM	X	X
Lymphknoten (befallen/ untersucht)	--	X
R-Klassifikation	--	X
Histologie	X	X
Histologie, Bemerkung	(X)	X
Grading	X	X
Rezeptoren	--	X [M]
Lokalisation	(X)	(X)
Seitenlokalisation	X	X
Therapie	--	X
Therapie, Datum	--	X
OPS-301 (Operationenschlüssel)	--	X
Ort	--	X
Therapie, Bemerkung	--	(X)
Therapieanlass	--	X
Komplikationen/ Nebenwirkungen	--	X
Rezidiv= Rückfall (Lokalrezidiv/ Metastasen)	--	X
Dokumentationsstand	X	X

[M] Mammakarzinom() fakultativ

Abb. 2.4. Übersicht über die Kurz- und Langdokumentation am OSP [OSP3]

Strukturiert sind die Daten durch die Einteilung in Stammdaten, Ersterhebung und Folgeerhebungen (Abb. 2.5.). Dabei werden die Stammdaten je Patient einmal erfasst. Je Primärtumor (ein Patient kann mehrere voneinander unabhängige Tumore haben) wird eine separate Ersterhebung mit den Daten zur Anamnese und Diagnose angelegt. Zu jedem Ersterhebungsbogen können mehrere Folgeerhebungen (Behandlungsbögen, Nachsorgebögen, Abschlussbogen...) gehören.

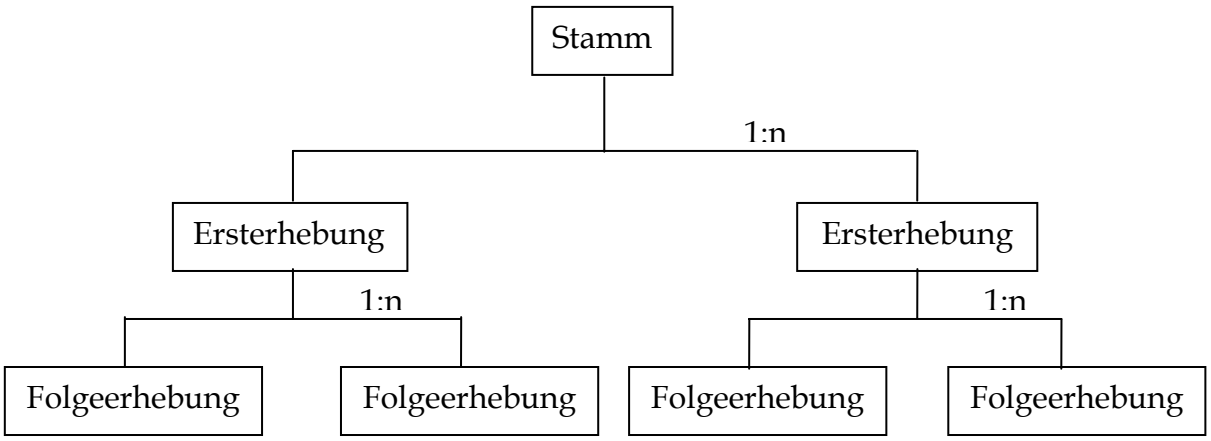

Abb.2.5. Struktur der Dokumentation [OSP4]

Zu den Aufgaben der Tumordokumentation am OSP Stuttgart gehören u.a.:

- Erfassung und Speicherung aller für die Behandlung und Versorgung eines Patienten notwendigen Krankheitsdaten über einen Zeitraum von mindestens 10 Jahren. Es gibt drei Behandlungsbögen (medizinisch, operativ und strahlentherapeutisch), einen Verlaufsbogen und einen Abschlussbogen.
- Sicherung der schnellen Verfügbarkeit und der Vollständigkeit der gespeicherten Daten für den behandelnden Arzt während der Therapie und der Nachsorge.
- Übersichtliche Verlaufsdarstellung
- Durchführung und Unterstützung der Qualitätssicherung

3. Bisheriges System und Caché

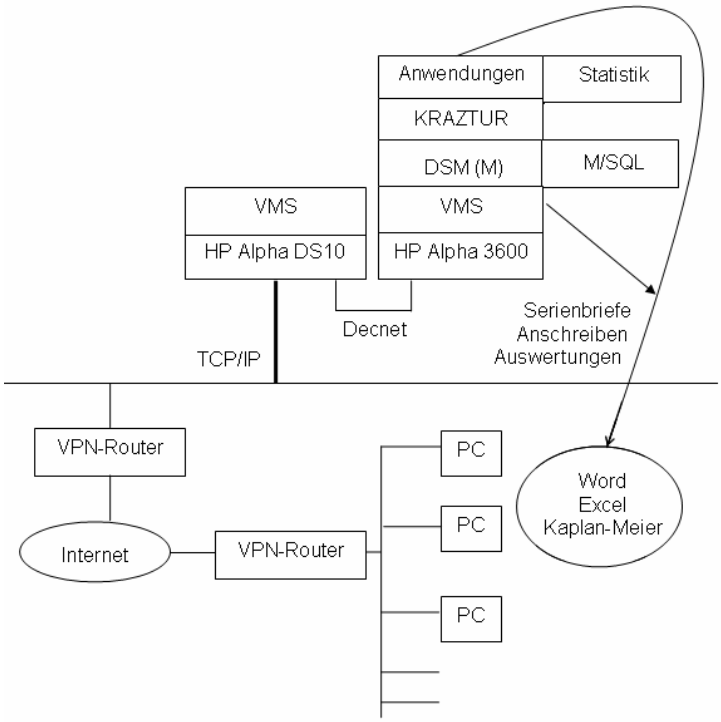

Abb. 3.1.a) Istzustand des Systems am OSP

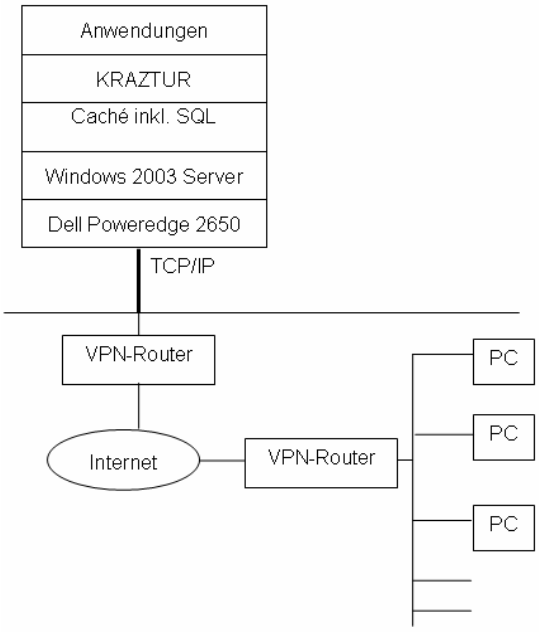

Abb. 3.1. b) Sollzustand des Systems am OSP

Zurzeit läuft die Datenbank auf HP Alpha 3600 unter dem Betriebssystem VMS (Abb. 3.1.a). Da dieser Rechner das TCP/IP-Protokoll nicht beherrscht, ist er über den HP Alpha DS10 mit dem Netzwerk verbunden. Auf dem VMS-Betriebssystem aufbauend ist die Datenbank DSM (M), früher MUMPS genannt, installiert. Mit Hilfe von M/SQL kann man die Datenbank auswerten. Der Datenbankgenerator KRAZTUR steht für die Funktionalität der Datenbank mit seinen Anwendungen und Statistikprogrammen. Um Serienbriefe, Anschreiben oder ähnliches zu verfassen, werden die Daten aus der Datenbank exportiert und zu den PCs portiert. Dort können sie in Word, Excel oder andere Programme integriert, bearbeitet, ausgewertet und ausgedruckt werden. Die PCs sind über einen VPN-Router mit dem Internet verbunden, genauso wie der HP Alpha DS10 und der zukünftige Datenbankserver Dell Poweredge 2650 (Abb. 3.1.b). Auf diesem Server ist das Betriebssystem Windows 2003 Server installiert und darauf die

Caché-Datenbank mit integriertem SQL, dem Datenbankgenerator KRAZTUR mit der gleichen Funktionalität wie im alten System und den Anwendungen, die eine neue Oberfläche bekommen sollen. Zurzeit existieren beide Systeme. Die Dokumentare arbeiten bis jetzt allerdings nur mit dem alten System, da die Rechtevergabe bei Caché noch nicht zufrieden stellend gelöst ist. Wenn die Probleme mit Caché vollständig behoben sind, müssen die aktuellen Datenbankinhalte nochmals nach Caché portiert werden. Dann kann das alte System vom Netz genommen werden. Bis die neuen graphischen Anwendungen fertig gestellt sind, wird auch auf Caché noch mit den zeichenorientierten Anwendungen von KRAZTUR gearbeitet.

3.1. MUMPS

Der Name steht für **M**assachusetts General Hospital **U**tility **M**ulti-**P**rogramming **S**ystem. MUMPS [WOE] [DAT] [MUG] ist eine im Bereich der medizinischen Informatik entstandene standardisierte Programmiersprache mit integrierter Datenhaltung, die die besonderen Bedürfnisse der Datenverarbeitung in der Medizin erfüllt: leistungsfähige Zeichenkettenverarbeitung und Datenhaltung. Durch seine Eigenschaften sowohl als Programmiersprache als auch als Datenbankensystem ist MUMPS besonders für interaktive Datenbankanwendungen geeignet. Diese offene Datenverarbeitungstechnologie ist ein ISO-Standard und besteht aus Programmiersprache, Datenspeicherungs- und Retrievalsystem, Transaktionsmonitor (multiuser und multitasking), Dialogmonitor, Netzwerkmanagement und Grafik-Interface. MUMPS ist auf viele Rechnerplattformen portiert und hat sich weltweit als herstellerunabhängige Datenverarbeitungstechnologie bewährt. Aufbauend auf den neuesten Erkenntnissen aus Informatik-Wissenschaft, Technik und Anwendung wird MUMPS laufend weiterentwickelt.
Neben den umfassenden Funktionen zur Stringverarbeitung stellt ein logisch konsistentes Variablenkonzept den Kern von MUMPS dar. Für den Programmierer ist nur die Zeichenkette variabler Länge als einziger Variablentyp sichtbar. Intern wird entweder ein logisches oder arithmetisches Datenformat oder der String gewählt, abhängig vom Typ der Daten. Dies führt zu leistungsfähigen und knapp formulierten Ausdrücken, da unterschiedliche Datentypen frei in Ausdrücken gemischt werden. Die Definition der Variablen erfolgt automatisch bei der ersten Zuweisung, wobei sie auch einen Index bekommen können. Dabei werden nur die Teile der indizierten Variablen

angelegt, denen ein Wert zugewiesen ist. Ein dynamisches Wachsen der Struktur ist möglich.
Wird MUMPS gestartet, wird aus einem PC ein Rechner mit einem multiuser, multitasking Betriebssystem, da MUMPS nicht nur die Datenbank sondern gleichzeitig auch sein eigenes Betriebssystem ist, das seinen Speicherplatz selbst verwaltet. Damit können an einem Rechner, an dem sonst nur eine Person arbeiten kann, viele Personen mit ihren Terminals arbeiten.
Die Vorteile von MUMPS bestehen also in einer leistungsfähigen, dynamischen Speicherverwaltung mit guter Speicherplatzausnutzung, einer hohen Flexibilität der Datenspeicherung und der Programme durch die jederzeitige Erweiterungsmöglichkeit des Codes und in der Möglichkeit des einfachen Einfügens von Programmcode während der Laufzeit. Desweiteren ist die hohe Portabilität durch den Standard und die Verfügbarkeit von MUMPS vom PC bis zum Main-Frame vorteilhaft, wie auch die guten Verarbeitungsmöglichkeiten von nichtnumerischen Daten, die hohe Leistungsausbeute aus dem Gesamtsystem und die leichte Erlernbarkeit.
Nachteilig wirkt sich die kompakte Schreibweise aus, da dies zu schwer lesbarem Programmcode führt. Durch fehlende Strukturierungsmöglichkeiten, wird dieses Problem noch verstärkt. Da auch die Dokumentation des Programmcodes fehlt, führt dies zu langwierigen Nachforschungen, bis die Programmlogik durchschaut ist und diese in das neue graphische Interface eingebaut werden kann.

3.2. KRAZTUR

KRAZTUR [ELK] [OSP2] [OSP4] ist das Akronym für **K**lein**R**echnergestütztes **a**llgemeines Dokumentationssystem mit **z**usätzlichen **T**extverarbeitungs- **u**nd **R**etrievalfunktionen. Es wurde 1977 von Dr. Karl-Heinz Ellsässer, dem Leiter der EDV-Zentrale und Tumordokumentation an der Universitätsklinik Heidelberg, in Zusammenarbeit mit dem Krebsforschungszentrum entwickelt [BVM]. Es ist ein Programmgenerator, mit dem die Funktionen des OSP selber programmiert werden können. So standen nicht alle Funktionen von Anfang an zur Verfügung. Das System konnte beliebig erweitert werden. Als Programmiersprache, auf der der Generator aufgebaut ist, benutzt KRAZTUR MUMPS, heute M genannt.
Für die Definition des Benutzerprofils, für die Erstellung einer Datenbank anhand einer Datenbankbeschreibung, für den Aufbau von Dialog- und Druckfunktionen und um codierte Merkmale anzulegen, wurden verschiedene Funktionen entwickelt.

Seit Einführung von KRAZTUR am OSP wurde die Funktionalität ständig erweitert und die Benutzerfreundlichkeit verbessert. So wurden unter anderem Bildschirm-Menüs für eine vereinfachte Benutzerführung (Abb. 3.2. - Abb. 3.3.) realisiert, Statistik- und Graphikpakete angebunden, ein Programm zur computerunterstützten Erstellung von Chemotherapieplänen integriert, Druckfunktionen eingebaut, ein Hintergrundmanagerprogramm zur besseren Ausnutzung der Rechenkapazität eingearbeitet, verschiedene Schnittstellen zu Betriebssystemen, Standard-Programmen wie Word oder Excel und anderen Rechnern integriert und ein einheitliches KRAZTUR-System mit anderen Benutzern festgelegt. Um auf die Daten mittels M/SQL zugreifen zu können wurde die Datenstrukturen angepasst. M/SQL ist eine auf SQL basierende Abfragesprache. Vor der Einführung dieser Sprache und der dazugehörigen Strukturen konnte nur mittels einer Retrievalfunktion abgefragt werden. Dabei bestand allerdings das Problem, dass nicht bogenübergreifend nach Daten gesucht werden konnte, also z.B. nur in den Ersterhebungsbögen oder Operationsbögen...

Abb. 3.2. jetziges Datenbank-Hauptmenü (KRAZTUR)

```
Kein Patient ausgewählt                                                  >ZEN

                    Inhaltsverzeichnis Krankenakte

 <1>  Patient auswählen              <9> Therapieplan erstellen

 <2>  Anzeige eingegeb. Aufenthalte <10> Übersicht Krankheitsverlauf

 <3>  Verlaufsinformation (kurz)    <11> Übersicht Therapien

 <4>  Verlaufsinformation (lang)    <12> Übersicht Therapie/Rezidive
      (nur Erst u. OP)
                                   <13> Pat.auswahl ohne Geb.dat.
 <5>  verabreichte Chemotherapie
                                   <14> Übersicht Rezidive
 <6>  Tumorparameter einzeln

 <7>  Tumorparameter u. Th.effekt     <z> zurück Datenbank-Menü

 <8>  Tu.parameter u. Chemotherapie   <e> Ende Datenbank

                        Ihre Wahl

Hilfe mit ?
  1(022,042)
```

Abb. 3.3. Inhaltsverzeichnis der Krankenakten (KRAZTUR)

Speicherungstechnik

Für die Stammdaten und jeden Erhebungsbogen eines Patienten wird in der Datenbank ein Datensatz mit intern fortlaufender Nummer, der Datensatznummer erstellt. Alle Daten werden im Global ^OP unter zwei Indizes, der Datensatznummer und dem Variablennamen gespeichert. Ein Global ist in MUMPS die Bezeichnung für eine globale Variable, die permanent gespeichert wird.

In KRAZTUR wird zwischen verschiedenen Speicherungsformen unterschieden. In einem einfachen Feld werden Merkmale gespeichert, von denen maximal eine Ausprägung pro Datensatz besteht. In der Merkmalsbeschreibung sind für jedes Einfachfeld der Variablenname und die Feldnummer festgelegt. Unter einem Variablennamen können mehrere Felder abgespeichert werden, die durch eine geschweifte öffnende Klammer getrennt werden. Die Stammdaten und der Ersterhebungsbogen werden zum Beispiel unter den Datensatznummern 100 und 101 wie folgt abgespeichert:

Stammdaten	^OP(100, ''ID“)= ''Nachname{Vorname{Geburtsdatum{...“
	...
Ersterhebung	^OP(101, ''TUM“) = ''Diagnose{Diagnosedatum{Klassifikation...“
	^OP(101, ''ZUWEIS“)= ''Zuweisungsgrund{Anlaß für Arzt-besuch...“
	...

Die zweite Speicherungsform ist das Wiederholmerkmal, bei dem mehrere Ausprägungen auftreten können. Alle Ausprägungen werden unter einem gemeinsamen Variablennamen gespeichert und in der Merkmalsbeschreibung wird als Feldnummer eine Zählvariable eingesetzt. Als letzte Speicherungstechnik gibt es noch die Wiederholungsgruppen, die aus mehreren logisch zusammenhängenden Merkmalen bestehen. Die Gruppen wiederholen sich als Ganzes, wobei für jedes Merkmal ein Feld reserviert ist und in der Merkmalsbeschreibung eine Zählvariable an den Variablennamen angehängt wird.
Jedem Patienten wird eine Codenummer zugewiesen, die der Identifikation dient und in jedem Datensatz abgespeichert wird.
In KRAZTUR gibt es drei verschiedene Datentypen: Text, Datum, Code, wobei die Abspeicherung der Textfelder dem Eingabetext entspricht, das Datum intern in der Form JJJJMMTT abgespeichert wird. Jede Ausprägung erhält beim Anlegen eines Codes ein Kürzel und einen zugehörigen Langtext. Den Kürzeln werden intern laufende Nummern zugewiesen, welche im Datenbestand abgespeichert werden.

3.3. Caché

Caché [KIW], [INT], [CACHE] ist meistens in Systeme eingebettet. Da die Datenbank dabei im Hintergrund arbeitet, ist bei den Anwendern kein Wissen über die Datenbank nötig, um die Applikation zu bedienen. Trotz des treuen Kundenstammes ist Caché von der Firma InterSystems noch nahezu unbekannt und kaum Literatur vorhanden. Die wenige Literatur stammt meist von InterSystems oder ihren Partnern. Dies kommt auch daher, da Caché an Universitäten noch kaum verbreitet ist und daher kein Bedarf an Lehrbüchern zu Caché besteht.
Angeboten wird Caché eingebettet in eine Vielzahl von Softwareprodukten, darunter medizinische Anwendungen, Entwicklungstools, ERP-Anwendungen, Online-Informationsprodukte für Abrechnungsstellen und sogar Logistiksoftware für das Verteidigungswesen. Etabliert hat sie sich allerdings hauptsächlich im medizinischen Bereich und zunehmend im Finanzwesen - beides Bereiche, die auf eine ausfallsichere, schnelle, fehlerfreie Datenbank

angewiesen sind. Verbreitung gefunden hat Caché vor allem in den USA, aber auch in Europa, Japan, Brasilien, Israel, Südafrika, Australien und Neuseeland. Laut einer Studie von IDC- Analyze the Future [IDC2] sind die postrelationalen Datenbanken die Technologie mit dem größten Zuwachs. Daher ist ein größerer Marktanteil und damit Bekanntheitsgrad von InterSystems (2003: 12,5% [IDC2] der embedded DBMS) für die Zukunft zu erwarten. Um diese Entwicklung zu unterstützen hat InterSystems das Projekt Caché Campus (http://www.intersystems.de/cache/communities/cache_campus/index.html) ins Leben gerufen. Bei diesem Projekt wird Caché an rund 200 Universitäten in 30 Ländern eingesetzt.

InterSystems lieferte zunächst Technologie im Zusammenhang mit MUMPS. 1997 kam Caché hinzu. Diese Datenbank verwendet die Storage-Engine der M-Technologie mit einigen Erweiterungen. Aufgrund dieser Vergangenheit gehören zum Kundenstamm viele, die die M-Technologie bereits seit Jahren einsetzen. Kunden wie der OSP Stuttgart. Dabei ist Caché unabhängig vom eingesetzten Betriebssystem. So läuft es unter anderem auf VMS, Unix oder Windows. Sogar die eingesetzten Betriebssysteme auf Client- und Serverseite können unterschiedlich sein. Dies kommt Firmen mit so heterogenen Strukturen wie dem OSP zugute. Eine Abstimmung zwischen den EDV-Abteilungen der angeschlossenen Kliniken über das Betriebssystem entfällt für diese Anwendung.

Caché wird vom Hersteller als postrelationale Datenbank bezeichnet. Zum einen ist damit die chronologische Einordnung des Datenbanksystems im Vergleich zu den relationalen Datenbanken gemeint, da die Grundlage für Caché später als das relationale Modell entwickelt wurde. Unter dem chronologischen Aspekt werden unter dem Begriff „postrelational“ alle Datenbanken, die nicht mehr auf normalisierten, flachen Tabellen basieren, also unter anderem verschachtelt-relationale, multidimensionale und objektorientierte Datenbanken zusammengefasst.

Der andere Gesichtspunkt für die Bezeichnung „postrelational“ liegt im Inhaltlichen. Die Anforderungen an eine Datenbank haben sich geändert. So sind heutzutage Anwendungen gefragt, die den komplexen Informationsfluss und vielfältige Arbeitsabläufe in Unternehmen unterstützen. Die Aufgabe der Entwickler besteht darin, diese makroskopischen Strukturen geeignet auf den Mikrokosmos der IT abzubilden. Davon sind auch die Modelle betroffen, die die Abläufe in Unternehmen beschreiben.

Die Terminologie „postrelational“ kann allerdings auch als PR verstanden werden, da sie weder das Produkt noch seine Funktionalität ganz beschreibt. Caché ist eine multi-dimensionale Datenbank, die die von InterSystems entwickelte Unified Data Architecture benutzt, um Caché sowohl als SQL-fähige relationale Datenbank (RDBMS) als auch als objektorientierte Datenbank (ODBMS) zu repräsentieren. Die Daten können als Objekte oder relationale Modelle verwendet werden (Abb. 3.4.). Dabei ist kein Mapping zwischen der Objekt-Welt und der relationalen Welt nötig. Die Daten werden intern durch das Datenbankmanagementsystem (DBMS) in multidimensionalen Arrays verwaltet. Dadurch ist ein schneller Zugriff mit einer Reihe von Front-Ends möglich. Die Vielfalt bei den Möglichkeiten des Datenzugriffs ermöglicht dem Benutzer, seine gewohnten Werkzeuge einzusetzen, was die Einarbeitungszeit wesentlich verkürzt. Das beste Ergebnis bei Anwendungen erhält man allerdings, wenn bis auf die graphische Benutzerschnittstelle alles in Caché gemacht wird. Dadurch werden die graphische Oberfläche und die dahinter steckende Logik voneinander getrennt. Beides kann so gesondert voneinander verändert werden. Dies wird auch durch die Objekteigenschaft der Kapselung begünstigt. Die Applikation betrachtet das Objekt als „Black Box“. Dadurch muss sie sich nicht um die interne Funktionalität eines Objektes kümmern. Sie benutzt nur die öffentlichen Eigenschaften und Methoden der Objekte. So können die internen Abläufe einzelner Objekte optimiert werden, ohne dass die Anwendung davon betroffen ist. Die Anwendung kann mit Hilfe der Kapselung mit der Komplexität der Daten fertig werden und sie gleichzeitig vor dem Benutzer verbergen.

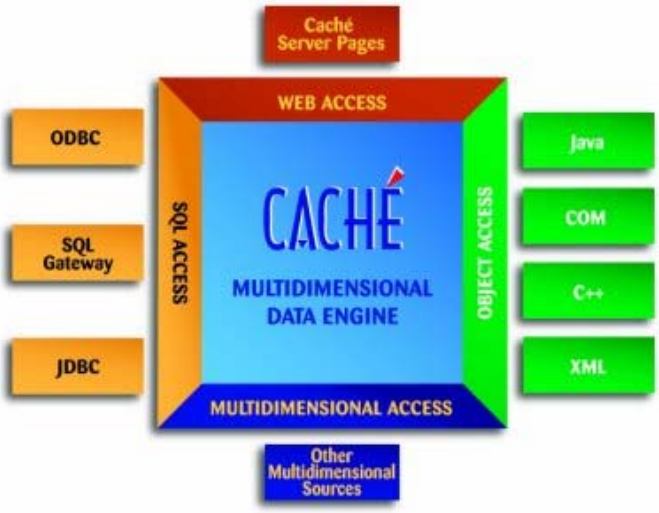

Abb. 3.4. Möglichkeiten des Datenzugriffs auf Caché [INT]

Ein weiterer Vorteil ist die umfassende Funktionalität. Denn hinter Caché steckt mehr als ein einfaches Datenmanagement. So können zum Beispiel XML-Daten über eine XML-Schnittstelle verarbeitet oder ausgegeben werden. Die Klassenmethoden sind direkt über den Caché-Server abrufbar. Auch die Erstellung kompletter Webanwendungen ist mittels einer eigenen Technologie,

den Caché Server Pages (CSP) realisierbar. Diese Technologie wird in dieser Studienarbeit in Kapitel 5.3. genauer betrachtet.
Die Schaltzentrale des Systems ist das Caché-Studio. Hier können die Struktur und das Verhalten von Caché-Objekten beschrieben, Caché Server Pages erstellt und Anwendungen getestet werden. Eine Caché-Klasse definiert die Datenstrukturen und die Logik, die in Caché abgelegt und gespeichert wird. Dabei sind die Eigenschaften (Properties) die Datenelemente einer Klasse und entsprechen den Spalten einer relationalen Tabelle. Die Methoden sind Funktionalitäten, die beim Aufruf eine Logik ausführen. Ein Caché-Objekt ist ein bestimmtes Vorkommen einer Klassendefinition. Mit Hilfe der Klassen sind in Caché eigene Datentypen beschreibbar, die sich aufeinander beziehen können. Bestehende Typen sind mittels Vererbung erweiterbar. Sobald eine neue Klasse angelegt wurde, ist auch eine relationale Sicht auf die Struktur verfügbar (Abb. 3.5.). Dies alles ist ebenso umgekehrt möglich, was vor allem für die Migration alter Datenbestände wichtig ist. Aus der relationalen Struktur sind über SQL Data Definition Language (DDL)-Statements die dazugehörigen Klassen erstellbar. Das Verhalten der Klassen kann mittels Scriptsprache oder Java beschrieben werden. Als Scriptsprache kann wahlweise Caché Basic und Caché ObjectScript verwendet werden. Beide haben intern nach der Kompilierung den gleichen Objektcode. Sie werden in ausführbaren Caché-Code übersetzt und laufen in der Run-Time-Umgebung von Caché ab. Die in Java geschriebenen Methoden werden dagegen vom Caché-Java-Binding zusammen mit der Klassendefinition an Java exportiert und laufen dann auf dem Client innerhalb der Java Virtual Machine ab.

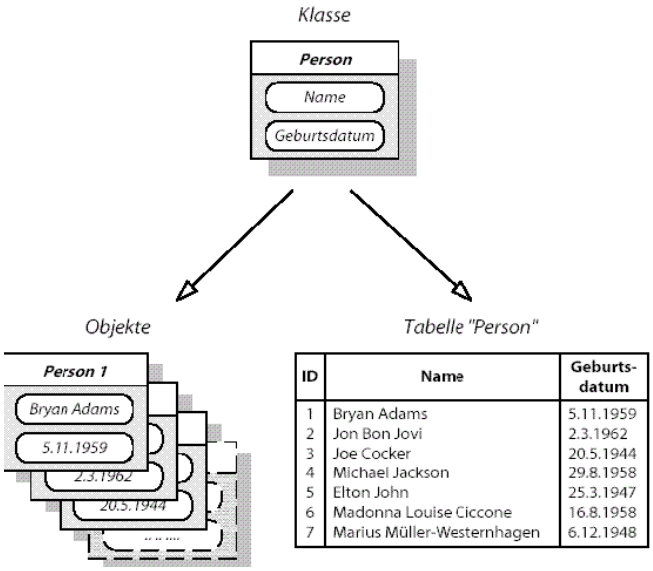

Abb. 3.5. Abbildung einer Klasse als Objekt und Tabelle [KIW]

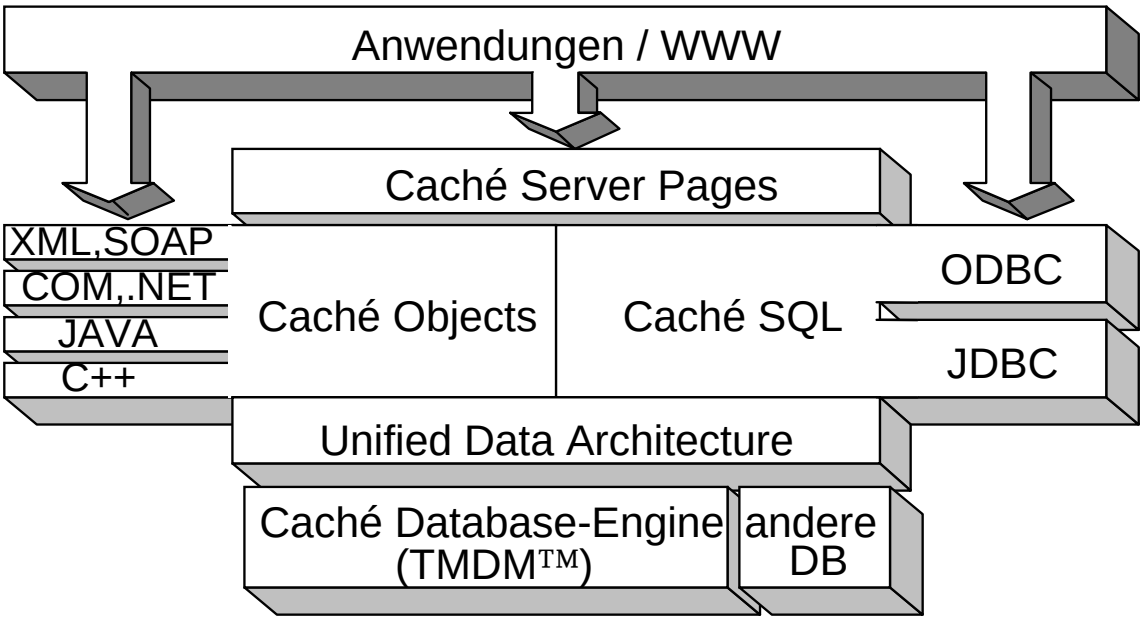

Abb. 3.6. Systemarchitektur von Caché [KIW]

Die Unified Data Architecture (Abb. 3.6.) bietet eine einheitliche Beschreibungsschicht für Objekte und Tabellen. Beide werden direkt in eine transaktionsorientierte, multidimensionale Datenbank-Engine (Abb. 3.6.) abgebildet. Dort werden die reich strukturierten Daten effizient und kompakt in multidimensionalen Arrays (Sparse Arrays) gespeichert. Die Datenlänge ist dabei variabel. Mit dieser Speichertechnik kann der Festplatten-I/O bei Datenzugriff und Aktualisierung reduziert werden.
Die multidimensionalen Strukturen werden in Caché Globals genannt. Dort können Daten mit einer beliebigen Anzahl von Subscripts gespeichert werden, wobei die Subscripts nicht an Typen gebunden sind.

^OP(100, „ID“) = „Nachname{Vorname{Geburtsdatum{...“
^OP(101, „TUM“) = „Diagnose{Diagnosedatum{Klassifikation{...“

Gespeichert werden die Daten und der Code in einer Datei mit dem Namen CACHE.DAT in einem bestimmten Verzeichnis im Dateisystem. Diese Datei entspricht auf Systemebene der Datenbank. Innerhalb der Datei müssen die Globalnamen eindeutig sein, zwischen unterschiedlichen Dateien spielt dies keine Rolle. Wo diese Dateien gespeichert sind, ist für den Caché-Prozess ohne Bedeutung. Zur Adressierung werden Namespaces verwendet. Dies sind logische Zuordnungen zwischen Globals, Code und den einzelnen Datenbanken. Werden diese verschoben, wird nur die Zuordnung im Namespace geändert. Die Daten, die zu einem Namespace gehören, sind alle in

einer Datenbank gespeichert, allerdings können unterschiedliche Datenbanken zu einem Namespace zusammengefasst werden.
Nach der Speicherung der Daten kann mit den drei Möglichkeiten: SQL, Objektzugriff und multidimensional auf die Daten zugegriffen werden, die Beste für die Anforderungen kann gewählt werden. Durch den Caché Objects-Server und den Caché-SQL-Server sind die nötigen Schnittstellen für relationale und objektorientierte Anwendungen vorhanden. Das Dilemma entweder SQL oder Objekte ist damit beseitigt. Es können die Vorteile der Objekttechnologie genutzt werden, ohne eine komplett neue objektorientierte Datenbank mit ihren Anwendungen entwerfen zu müssen. Zudem wurde der Objektzugriff um ein objektorientiertes SQL erweitert. Damit kann die mühevolle Join-Syntax weitgehend umgangen werden.
Positiv auf die Performance der Datenbank wirkt sich auch aus, dass bei Transaktionen nicht die kompletten Datenblöcke gesperrt werden, sondern nur eine Sperrung auf logischer Ebene erfolgt. Dies ermöglicht einen reibungslosen Ablauf zeitgleicher Zugriffe auf die Datenbank, da es so zu weniger Konflikten zwischen konkurrierenden Prozessen kommt. Der durchgehende Datenbankzugriff ist auch während Backups gewährleistet. Dies ist durch die Unterstützung von vollständigen, inkrementellen und kumulativ-inkrementellen Backups möglich. Die Backups können während des laufenden Betriebes durchgeführt werden. Auch Quellcode-Änderungen können mit Vorsicht bei laufendem System vorgenommen werden. Dabei werden die geänderten Quellroutinen auf den Daten-Server geladen. Dort werden sie automatisch kompiliert und die Applikationsserver benachrichtigt, um die überarbeiteten Routinen neu zu laden. Ein Linken ist nicht erforderlich. Da der Code von Routinen in der Datenbank gespeichert ist, können sie dynamisch im gesamten Netzwerk aufgerufen werden und müssen nicht auf jedem Rechner installiert werden. Dies erleichtert die Änderung der Routinen erheblich.

Der Caché-Applikationsserver

Caché ist mehr als ein DBMS. Es verfügt über einen Applikationsserver (Abb. 3.7.) mit modernsten objektorientierten Programmiereigenschaften, Integrationsmechanismen für weitere Technologien und eine Laufzeitumgebung, die Caché Virtual Machine. Diese unterstützt die zwei Scriptsprachen Caché ObjectScript und Basic. Caché ObjectScript ist eine leistungsstarke Sprache mit flexiblen Datenstrukturen, die allerdings den Nachteil hat, dass sie an vieles erinnert, was Programmierer kennen, aber mit keiner Sprache identisch ist. Dem versucht InterSystems entgegenzuwirken, indem es die Sprache stärker an gewohntes anlehnt und die Akzeptanz damit erhöht. Basic dagegen ist dem Visual Basic Script sehr ähnlich. Für den Zugriff auf die multidimensionalen Arrays ist die Sprache gegenüber Visual Basic

erweitert worden. Der Code für die Scriptsprachen wird in der Datenbank gespeichert und kann während des Betriebes geändert werden, worauf eine automatische Weiterleitung der Änderungen an den Applikationsserver erfolgt.

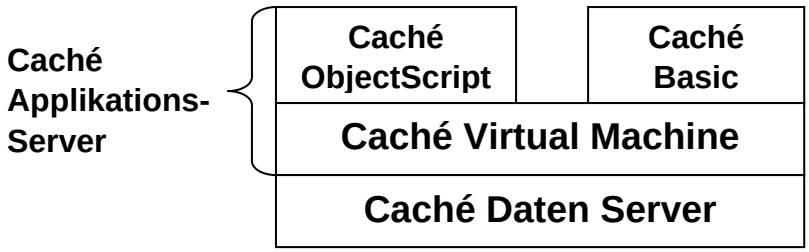

Abb. 3.7 Caché Applikations-Server [INT]

Der Applikationsserver kann sowohl auf die multidimensionalen Daten-Server von Caché als auch auf relationale Datenbanken zugreifen. Mit JDBC und ODBC ist auch ein SQL-Zugriff möglich.
Durch Konnektivitäts-Software mit Caching auf der Client-Seite wird der Zugriff auf alle gängigen Technologien wie Java, C++, COM, .NET, Visual Basic und Delphi gewährleistet. Durch die automatische Vernetzung zwischen dem Client und dem Applikationsserver durch Caché ist die Entwicklung einer Client-Server-Applikation möglich. Die Kompatibilität zu SOAP und XML gestattet eine Anwendung mittels Web Service-Technologie.
Die dritte Möglichkeit der Umsetzung einer Anwendung wird durch den Caché-Applikationsserver erst ermöglicht. Die Technologie der Caché Server Pages führt zu hochperformanten Webanwendungen, deren Erstellung durch InterSystems supportet und nahegelegt wird

Abschließend kann man sagen, dass Caché ein komplettes DBMS ist, das sich durch die Eigenschaften Performance, Skalierbarkeit, Sicherheit, Objektorientierung und eine relationale Sicht auf die Daten auszeichnet und dabei wenig Administration benötigt. Durch seine noch nicht flächendeckende Verbreitung über alle Bereiche, in denen Datenbanken eingesetzt werden, fehlt Caché noch ein wenig Akzeptanz in der IT-Welt.

4. Überführung der Daten

In den CACHE.DAT Dateien sind nicht nur die reinen Patientendaten, sondern auch die Routinen und Steuerungsdaten gespeichert. Mit den Steuerroutinen I0 und K2 wurde in KRAZTUR z.B. die Benutzerführung gesteuert. Da diese sich in einer graphischen Oberfläche komplett von der in KRAZTUR unterscheidet, werden diese Routinen nicht mehr gebraucht.

4.1. Die Portierung der Daten von MUMPS nach Caché

Da die M-Datenbank hierarchisch ist, war die Portierung in die Klassen kein Problem.

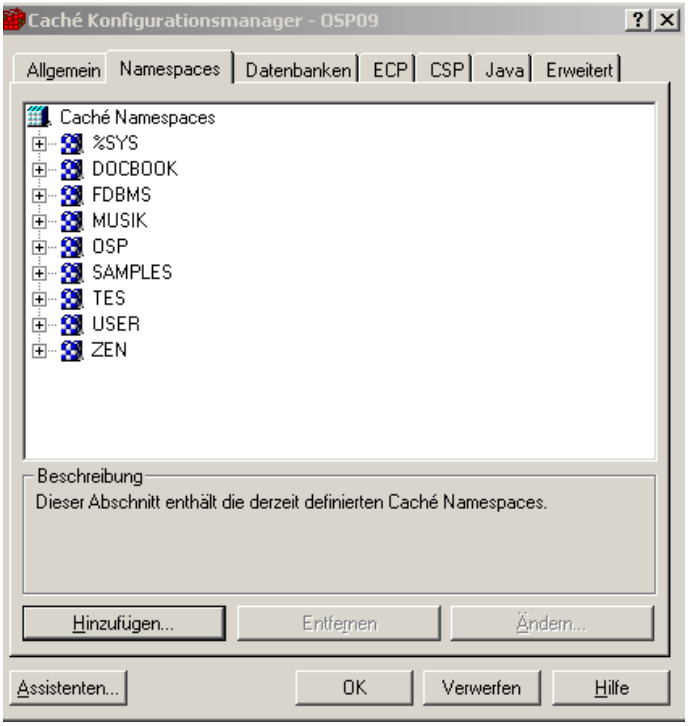

Abb.4.1. Konfigurationsmanager

Bevor damit angefangen werden konnte, mussten bei Caché mit Hilfe des Konfigurationsmanagers (Abb. 4.1.) neue Namespaces und Datenbanken angelegt werden. Diese haben dieselbe Struktur wie davor in der Mumps-Datenbank.

Angelegt wurden die Namespaces OSP, ZEN und TES und die jeweils dazugehörigen Datenbanken. In der Datenbank OSP sind die allgemeinen Daten gespeichert, wie z.B. die Namen der Krankenhäuser oder die Schlüssel für die Diagnosen und Therapien. Die Patientendaten sind in ZEN gespeichert. TES ist eine Testdatenbank, in der neue Routinen an Testdaten getestet werden können. In jeder Datenbank sind die dazugehörigen Routinen gespeichert. Auch diese mussten in Caché importiert werden.

Nachdem die Namespaces und Datenbanken in Caché angelegt waren, konnte mit der Portierung der Daten und Routinen begonnen werden. Dazu mussten die in M gespeicherten Daten über ein M-Systemprogramm in eine Datei geschrieben werden. Nachdem der Systemadministrator von KRAZTUR nach VMS gewechselt ist, kann er über den Befehl „mumps osp“ direkt auf der M-Datenbank OSP arbeiten. Dort können mit dem Befehl „%GTO“ (Global Transfer Output) die Globals ausgelesen und in eine Datei gespeichert werden.

Zuerst wird der Systemadministrator nach der Zieldatei „*.dat“ gefragt. Danach kann er eine Beschreibung für die Datei angeben und die Globals der Reihe nach oder alle auswählen, die er in die Datei speichern will. Hat er seine Auswahl beendet, erfolgt das Schreiben in die angegebene Datei. Um die kompletten Daten nach Caché zu portieren, müssen alle Globals aus ZEN und OSP in eine Datei ausgelesen werden. Dazwischen muss natürlich die M-Datenbank gewechselt und nach dem Auslesen verlassen werden.
Auf die gleiche Art und Weise wie die Globals können die Routinen mit Hilfe des Befehls „%RS“ (Routine Save) ausgelesen werden. Die Dateien liegen jetzt alle in einem Verzeichnis auf VMS. Da von Windows aus auf dieses Verzeichnis nicht zugegriffen werden kann, weil es nicht freigegeben ist, müssen auf VMS die Dateien in ein Verzeichnis DKA300:[000000] kopiert werden. Dieses Verzeichnis ist das einzige VMS-Verzeichnis, auf das von Windows zugegriffen werden kann. Das Kopieren erfolgt mit dem Befehl

„copy *dateiname*.dat *Servername*“*Benutzername Passwort*“::*Device-Name* :[*Verzeichnis*]*.*/log“

Die Endung „/log“ bewirkt dabei, dass der Vorgang protokolliert wird. Mit Hilfe dieses Protokolls kann später nach eventuell aufgetretenen Fehlern hier nach einer Ursache gesucht, beziehungsweise die Ursache bei der Datenübertragung in die neue Datei ausgeschlossen werden. Nun können mit dem Programm „FTP per Drag-und-Drop-Client“ die Dateien auf die lokale Festplatte portiert werden.

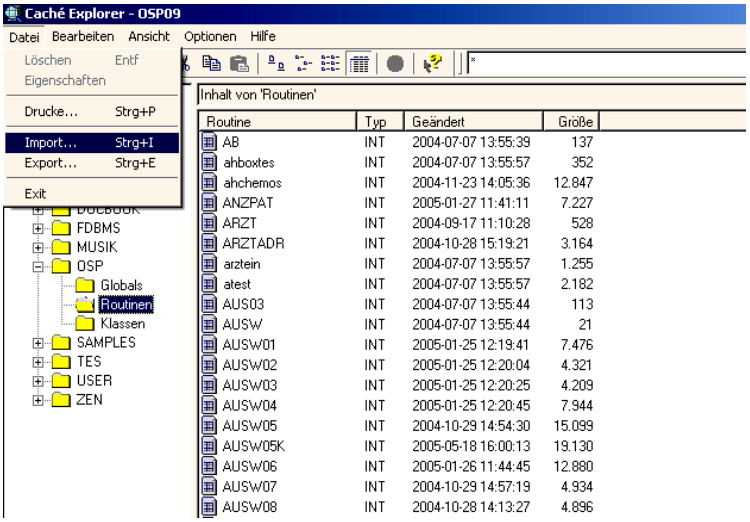

Abb. 4.2. Caché Explorer zum Importieren der Daten

Von der lokalen Festplatte werden die Dateien über eine direkte Verbindung auf den Caché-Server kopiert. Es ist nur ein einziges Verzeichnis freigegeben, um Daten von außen auf den Server zu kopieren. Allerdings ist der Zugriff auf dieses Verzeichnis aus Sicherheitsgründen nur mit speziellen Rechten möglich. Jetzt liegen die Daten schon auf dem richtigen Server und müssen nur noch in die Datenbank importiert werden. Die Dateien mit den Routinen müssen in die Routinen, die mit den Globals in die Globals der jeweiligen Datenbank

importiert werden. Für den Import der Globals muss unter Einstellungen noch das Importformat auf DSM-11 und bei der Importierung der Routinen der Sprachmodus auf DSM-11 und das Importformat auf DSM festgelegt werden. Während des Imports werden die Dateien kontrolliert und kompiliert. Wenn hier keine Fehler festgestellt worden sind, sind die Daten nach Caché portiert worden.

4.2. Probleme bei der Überführung der Daten

Bei der Portierung der Globals kam es zu keinen Problemen. Bei den Steuerroutinen zur Benutzerführung traten allerdings drei Arten von Problemen auf.

Zum einen spielte es in M keine Rolle, wenn zwei Unterprogramme denselben Namen hatten. Caché akzeptierte dies dagegen nicht. Daher mussten einige Unterprogramme umbenannt werden.

Ein weiteres Problem stellten Befehle dar, die so in Caché nicht erkannt wurden oder eine andere Syntax hatten. Ein Beispiel dafür ist die Variablenzuweisung, die in M „:(Y=y+2, X=x)“ lautete. In Caché muss dieser Befehl durch „set y=y+2 set x=x“ ersetzt werden. Ein weiterer Befehl, der in Caché eine andere Syntax hatte, war der Befehl für die Bildschirmausgabe. In M reichte hierfür ein W (als Abkürzung für write, wobei auch der Name ausgeschrieben werden konnte). In Caché musste dies durch „write @mv W“ ersetzt werden. Dies war mit Hilfe eines Scriptprogramms, das die Firma InterSystems programmierte, leicht möglich.

Das größte Problem stellte die Anpassung der Koordinaten dar. In M wurde eine Variable, z.B. „y“ für den Ausgabestart auf dem Bildschirm gewählt. Die nächsten Zeilen („Y“) auf dieser Seite hingen nun alle von dieser Variablen ab (Abb. 4.3. Zeile 4-7). Da das von InterSystems geschriebene Scriptprogramm nicht zwischen Groß- und Kleinschreibung unterschied, hing die nächste Zeile der Ausgabe nach der Ausführung des Scriptprogramms immer nur von der Vorgänger-Zeile ab, der Abstand der Zeilen war dann aber nicht mehr konstant. Dies führte allerdings zu großen Anpassungsproblemen. Hier war viel „Handarbeit“ nötig, bis das ursprüngliche Layout wieder hergestellt war. (Abb. 4.4. Zeile 4-7).

```
Zeile 1: ;U 0:(Y=y,X=28) W @inv," Datenbank-Hauptmenu ",@aus,@fett
Zeile 2: U 0:(Y=y,X=28) W %T("Ron")," Datenbank-Hauptmenu",%T("Roff"),%T("Hon")
Zeile 3: S x=17
Zeile 4: U 0:(Y=y+2,X=x) W "<1>   Patienten erfassen/ändern        "
Zeile 5: U 0:(Y=y+4,X=x) W "<2>   Überweisung (alt)              "
Zeile 6: U 0:(Y=y+6,X=x) W "<3>   Aufnahmekontrolle              "
Zeile 7: U 0:(Y=y+8,X=x) W "<4>   Krankenakte                    "
```

Abb. 4.3. Ausschnitt aus der Routine MHAUPTM aus M

```
Zeile1: ;U 0:(Y=y,X=28) W @inv," Datenbank-Hauptmenü ",@aus,@fett
Zeile2: U 0 set y=%T("Fline")+2 set x=28 write @mv W %T("Ron"),"Datenbank-Zeile2b:
Hauptmenu",%T("Roff"),%T("Hon")
Zeile3: S x=17
Zeile 4: U 0 set y=y+2 set x=x write @mv W "<1> Patienten erfassen/ändern "
Zeile 5: //U 0 set y=y+2 set x=x write @mv W "<2> Überweisung (alt) "
Zeile 6: U 0 set y=y+4 set x=x write @mv W "<3> Aufnahmekontrolle "
Zeile 7: U 0 set y=y+2 set x=x write @mv W "<4> Krankenakte "
```

Abb.4.4. Ausschnitt aus der Caché-Routine MHAUPTM

Zusammenfassend kann man sagen, dass die Portierung der reinen Patientendaten vom alten System schnell ging, da Caché die Daten aus MUMPS leicht und schnell einbinden kann. Vorteilhaft für die Portierung der Daten nach Caché war, dass sie für die Einführung von M/SQL in ihrer Datenstruktur bereits soweit angepasst wurden, dass der Zugriff mit der SQL-ähnlichen Abfragesprache möglich wurde.

Die Anpassung der Routinen erforderte jedoch viel Handarbeit. Danach konnte die Dokumentationsanwendung mit der Caché-Datenbank mit der zeichenbasierten Dialoganwendung arbeiten. Ein Problem dabei waren allerdings die Benutzersichten auf die Daten. Bei Abfragen dürfen die Dokumentarinnen und Dokumentare nur die Daten der Patienten aus „ihrem" Krankenhaus sehen. Die anderen müssen für sie aus Datenschutzgründen verborgen bleiben. Die Benutzersicht einzuschränken ist nur durch eine CSP-Seite möglich, wobei im Hintergrund eine Routine läuft, die den Zugriff auf die Daten einschränkt, für die der eingeloggte Benutzer Zugriffsrechte hat.

Mit der Entwicklung dieser CSP-Seite und der im Hintergrund laufenden Routine wurde eine Firma beauftragt.

Wenn dieses Problem endgültig gelöst ist und die graphische Oberfläche fertig gestellt ist, werden die Steuerungsroutinen überflüssig. Die HTML-Oberfläche benötigt die Steuerung nicht mehr.

5. Möglichkeiten der Umsetzung

Durch seine Systemarchitektur bietet Caché vielfältige Möglichkeiten der Anwendungen. Für diese Studienarbeit sollen die Möglichkeiten einer Client/Server-Applikation, einer Realisierung mittels Web Services und die Erstellung der Anwendung mittels Caché Server Pages analysiert und evaluiert werden.

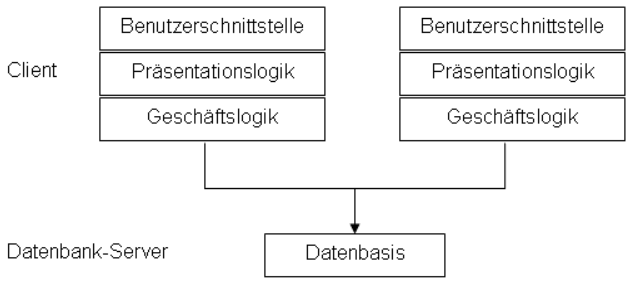

Abb.5.1. Zwei-Schichten-Architektur [IDM]

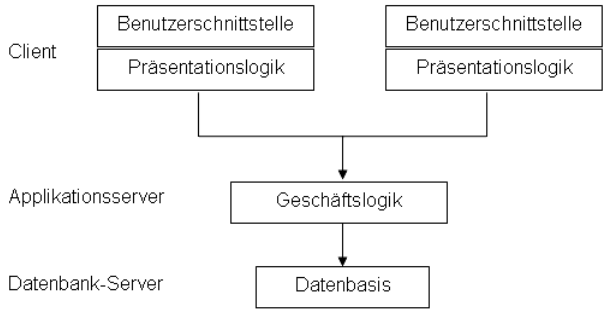

Abb. 5.2. Drei-Schichten-Architektur [IDM]

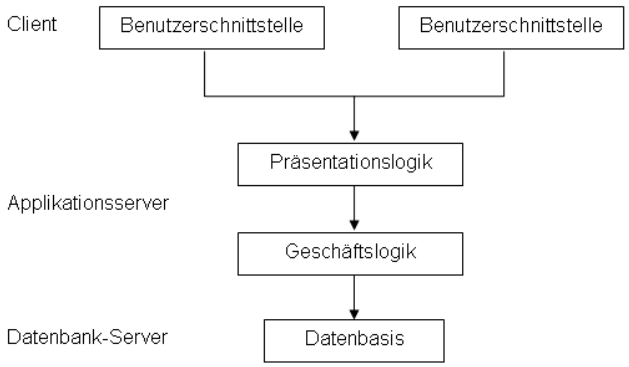

Abb. 5.3. Vier-Schichten-Architektur [IDM]

Da die drei Umsetzungsmöglichkeiten aus Server und Clients bestehen, sind alle drei Client/Server-Applikationen, allerdings mit unterschiedlich vielen Schichten [IDM]. Die klassische Client/Server-Lösung (Abb. 5.1.), die in Kapitel 5.1. behandelt wird, besteht aus zwei Schichten (im weiteren nur Client/Server genannt). Auf dem Server befindet sich nur die Datenbasis. Der Zugriff auf die Daten wird dabei von der Datenbank verwaltet. Die Client-Anwendung besteht aus der kompletten Präsentations- und der Geschäftslogik.

Die Web Services (Abb. 5.2.) haben eine Drei-Schichten-Architektur. Die Geschäftslogik ist nicht mehr auf dem Client untergebracht, sondern auf dem Applikationsserver. Auf die Daten wird hier von der Präsentationslogik auf dem Client über die Geschäftslogik zugegriffen. Die Datenbankverbindung wird vom Applikationsserver verwaltet.

Vier Schichten hat schließlich die Realisierung mittels Caché Server Pages (Abb. 5.3.). Hierbei ist auch die Präsentationslogik weg vom Client auf den

Applikationsserver verlegt worden. Auf dem Client ist nur noch ein Web-Browser nötig. Daher werden diese Clients auch Thin Clients genannt.
In den Kapiteln 5.1. bis 5.3. werden die drei Möglichkeiten mit ihren Anforderungen vorgestellt und die Vor- und Nachteile herausgearbeitet. In Kapitel 5.4. folgt das Fazit dieser Evaluierungen und damit die Begründung für die Entscheidung, mit Hilfe welcher Technologie die Anwendung schließlich realisiert wird.

5.1. Client/Server-Applikation

5.1.1. Analyse

Das Client/Server-Modell [TEK][NIK][ELE] (Abb. 5.4.) ist sozusagen aus der Not geboren. Zu Beginn des Computerzeitalters waren Computer sehr teuer und Speicherplatz sowie Peripherie-Geräte wie Drucker fast unbezahlbar. Durch den Aufbau eines Netzwerkes zwischen Computern können Ressourcen und Daten gemeinsam genutzt und Programme und Daten zentral gesteuert werden. Dadurch steigert sich die Leistung des Systems. Am Anfang dieser Technologie stand ein einfaches Netzwerk aus mehreren eigenständigen Computern. Darauf folgte ein Disk-Server, auf dem ein Betriebssystem installiert war. Dieses war für die Organisation der Zugriffe der Clients zuständig. Zu dieser Zeit konnten die Zugriffsrechte von einzelnen Computern schon eingeschränkt werden. Der Wartungsaufwand war allerdings riesig, da die Zuständigkeit für die Verwaltung bei den Clients lag. Daher wurde der Disk-Server vom File-Server abgelöst. Ab jetzt ist der Server für die Verwaltungsaufgaben zuständig.

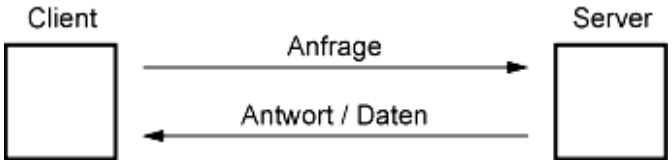

Abb. 5.4. Client/Server-Architektur [ELE]

> Ein System wird als Client/Server-System bezeichnet, wenn es aus mehreren Komponenten besteht, diese miteinander kommunizieren und eine Aufgabenverteilung zwischen ihnen stattfindet in der Form, dass eine Komponente (Client) Dienste von einer anderen Komponente (Server) anfordert, die diese Dienste erbringt.

Abb. 5.5. Definition von Client/Server-System von [TIW]

Soviel zur Geschichte der Client/Server-Architektur. Sie besteht also aus einem oder vielen Clients, die auf die Ressourcen des Servers zugreifen. Der Server verwaltet als Anbieter die Ressourcen zentral und stellt sie und die Funktionalität der Anwendungsprogramme den Clients zur Verfügung. Diese bieten die Benutzeroberfläche oder die Benutzerschnittstelle der Anwendung

an. Die dritte Komponente der Architektur ist die Kommunikationsverbindung zwischen dem Client und dem Server.
Heute ist aufgrund der Vorteile der Personal-Computer gegenüber den Terminals oftmals viel Rechenkapazität auf den Arbeitsplatzrechnern angesiedelt. Der Server bringt dann nur noch die Daten in das System ein. So ein zentraler Datenbankserver käme am OSP bei der Verwendung der Client/Server-Architektur für die Anwendung in Frage. Eine andere Möglichkeit im Zeitalter des Intranet wäre, den Browser als Thin Client einzusetzen und die Programmlogik auf einem Applikationsserver abzulegen.
Der Leitgedanke der Client/Server-Technologie besteht somit darin, die mehrfach verwendeten Dienste zu identifizieren und sie aus den Anwendungsprogrammen, in denen sie genutzt werden, herauszulösen. Danach werden sie dann einmal in allgemeingültiger Form implementiert und netzweit zur Verfügung gestellt. Hierfür wird ein Application Programming Interface (API) genutzt. Im Falle einer Client/Server-Datenbank stellt die API die SQL-Schnittstelle lokal zur Verfügung.

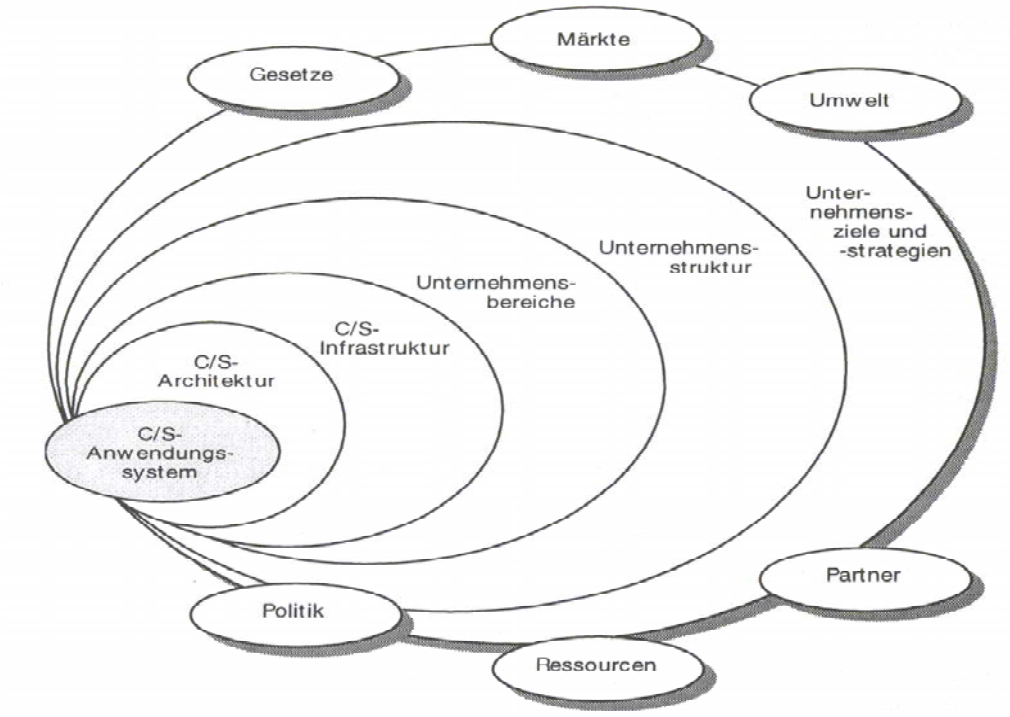

Abb. 5.6. Client/Server im Unternehmen [TIW]

Bei der Analyse dieser Möglichkeit der Umsetzung spielt das Zusammenwirken von Client/Server-Architektur und –Infrastruktur, aber auch die Unternehmensstruktur (Abb. 5.6.) eine Rolle. So müssen alle abgebildeten Schichten bei der Planung und Realisierung eines Client/Server-Systems berücksichtigt werden. Die Client/Server-Anwendungen spielen als Werkzeuge, mit denen die Dokumentare im OSP arbeiten, eine wichtige Rolle

im System. Die Infrastruktur besteht heutzutage meist aus einem lokalen Netzwerk mit eingebundenen PCs, auf denen die Dialogführung stattfindet. Dies wäre auch im OSP der Fall. Die Daten werden auf einem leistungsstarken Datenbankserver, dem Caché/Server gehalten. Die Aufgabe der Kommunikation wird von einem Mailserver erledigt, die netzweite Bereitstellung eines Dateisystems mit hoher Speicherkapazität und schneller Zugriffszeit ist Aufgabe des Fileservers. Als letzte Komponente steuert der Netzserver den Netzbetrieb und sorgt für dessen Betrieb.

5.1.2. Evaluation

Bevor das Client/Server-System in Betrieb genommen werden kann, sind ein genauer Entwurf und eine präzise Planung nötig. Wichtige Fragen in der Planung sind:

- Implementiert der OSP alles selbst oder beauftragt er damit eine Firma?
- Wie viel Funktionalität soll serverseitig implementiert werden?
- Was wird clientseitig installiert?

Bei Beantwortung dieser Fragen, fallen einem sofort die Hauptnachteile der Client/Server-Technologie für die Tumordokumentation am OSP auf.
Zum einen stehen von Caché keine fertigen Bausteine für eine Client/Server-Applikation zur Verfügung. So müsste alles, server- wie clientseitig, selbst entwickelt werden. Dadurch wird die Entwicklung der Anwendung sehr teuer. Es spielt dabei keine Rolle, ob die Applikation im Haus implementiert wird oder eine Firma damit beauftragt wird. Bei letzterem wären mit Kosten von mindestens 40000 bis 50000€ zu rechnen.

Das zweite Problem, das vor allem für den Betrieb der Anwendung große Nachteile nach sich ziehen würde, ist die clientseitige Installation. Da die Anwendung in allen Krankenhäusern, die dem OSP angeschlossen sind, genutzt wird und diese auf ganz Stuttgart verteilt sind, wäre die Verteilung der Software mit einem großen organisatorischen Aufwand verbunden. Die Software müsste auf allen Arbeitsplätzen installiert werden. Der gleiche Zeitaufwand müsste bei jedem Update aufgebracht werden. Da bis jetzt im Mittel circa einmal im Monat etwas an der Anwendung geändert wird und dies auch für die Zukunft zu erwarten ist, wäre der Arbeitsaufwand dafür erheblich. Erschwerend käme noch hinzu, dass die einzelnen Arbeitsplätze nicht dem OSP unterliegen, sondern die Dokumentare/ Dokumentarinnen bei den jeweiligen Krankenhäusern angestellt, die PCs Eigentum der Krankenhäuser sind und die Administration der Hardware, mit der die Dokumentare/ Dokumentarinnen arbeiten, den einzelnen EDV-Abteilungen unterliegt. Die PC's sind in die IT-Infrastruktur und die Sicherheitsstruktur der

jeweiligen Krankenhäuser eingebettet. Dadurch treten unterschiedliche Standards, Zugriffsrechte und Betriebssysteme auf.
So müsste vor Inbetriebnahme der neuen Anwendung eine derartige Abmachung zwischen den einzelnen EDV-Abteilungen getroffen werden, dass die Software auf den PCs installiert werden darf und gegebenenfalls von der EDV-Abteilung vor Ort auch gewartet wird. Für eine konsistente Dokumentation wäre es dringend erforderlich, dass alle Dokumentare/ Dokumentarinnen mit derselben Version der Anwendung arbeiten und dies ist eine fast unerreichbare Anforderung.
So kann das Problem mit der Wartung der Anwendung als K.o.-Kriterium angesehen werden, die Anwendung als Client/Server-Applikation zu realisieren ist nicht empfehlenswert.

5.2. Web Services

5.2.1. Analyse

Simple Object Access Protocol (SOAP) Web Services [KIW], [CACHE] basieren auf dem Kommunikationsformat XML (eXtensible Markup Language). Mithilfe dieser Standardsprache für den plattformunabhängigen Austausch von Daten werden bei SOAP Funktionsaufrufe codiert, wodurch sie zwischen beliebigen Systemen ausgetauscht werden können (Abb. 5.7.). Die gemeinsame Nutzung von Anwendungsfunktionalität ist sowohl über das Internet wie auch innerhalb einer Organisation oder eines Systems möglich. Das XML-Dokument, das auf dem SOAP-Protokoll basiert, wird über die Schnittstelle der Web Services geschickt, die in WSDL (Web Service Definition Language) geschrieben ist. Das XML-Dokument wird also als SOAP-Nachricht an einen Web-Server geschickt und fordert dort Web Services an. Die WSDL beschreibt die Web Services. Hier werden die Art und Anzahl der Parameter sowie die URL des Web Service spezifiziert. So können die Clients die erforderlichen Dienste per SOAP-Nachricht anfordern, wenn nötig mehrere Clients

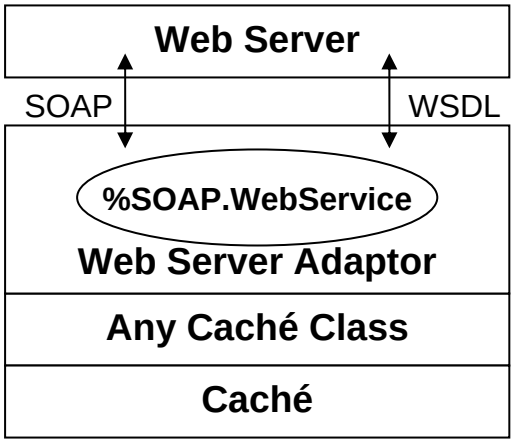

Abb. 5.7. Aufbau eines Web Service bei Caché [INT]

gleichzeitig. Dies ist vor allem vorteilhaft, wenn die Anwender der Applikation weit verteilt sind. Die Administration der Applikation wäre sonst sehr schwierig und teuer. Der Web Service kann dagegen zentral administriert werden. Daher ist der Web Service auch für Anwendungen geeignet, deren Funktionalität oft geändert wird, wie es bei der Anwendung vom OSP der Fall ist. Bei einer Änderung der Funktionalität muss nur der Web Service angepasst werden. Die Clients können unverändert bleiben, solange die in der WSDL beschriebene Schnittstelle beibehalten wird.

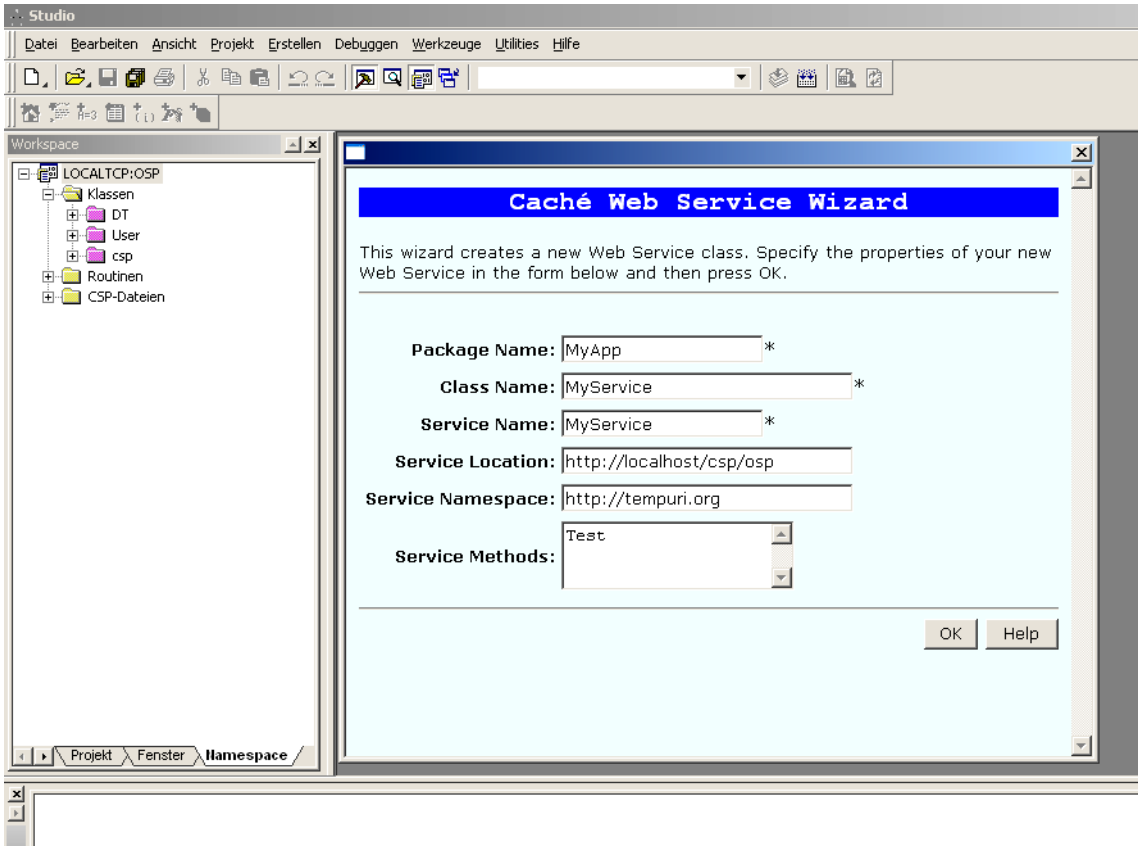

Abb. 5.8. Form Wizard im Caché Studio zum Erstellen eines Web Service

Zum Erstellen von Web Services stellt Caché im Studio einen Wizard (Abb. 5.8.) zur Verfügung. Mit diesem können alle Klassenmethoden, die SQL Stored Procedures und jede beliebige Anfrage als Web Service dargestellt werden. Dafür sind nur wenige Mausklicks nötig. Dasselbe Ergebnis ist zu erreichen, wenn hinter die Definition der Klassenmethode [WebMethod] angehängt wird. Beim Kompilieren der Klassen, die mit [WebMethod] markierte Methoden enthalten, werden für diese automatisch Klassen erzeugt, die die benötigten Operationen bereitstellen. Mit diesen können WSDL erzeugt und SOAP-Aufrufe decodiert werden. Die Aufrufe werden ausgeführt und das Ergebnis als SOAP-Nachricht zurückgeschickt.

Da SOAP als Kommunikationsprotokoll genutzt wird, kann der Zugriff auf den Web Service über viele Programmiersprachen und Plattformen erfolgen. Die einfachste Möglichkeit ist über das CSP-Gateway. Dabei kann eine Übersichtsseite des Web Services vom Studio aus im Web-Browser geöffnet werden, hierfür ist dann keine weitere Bibliothek nötig. Diese Möglichkeit muss aber sofort wieder eingeschränkt werden, da sie eigentlich nur zum Testen von Web Services dient. Der Test der Web Services ist also ohne Client Anwendung möglich. Für den Betrieb einer Anwendung, die auf Web Services basiert, ist allerdings sowohl eine Client-Anwendung wie auch ein richtiger Web Server nötig. Eingesetzt werden können hier bekannte Web-Server wie Apache oder Tomcat. Für die Client-Applikation werden viele Sprachen unterstützt. So ist eine Entwicklung mit Visual Basic oder .NET genauso möglich wie mit Java. Diese Sprachen unterstützen das Kommunikationsprotokoll SOAP und bieten dafür Toolkits oder Bibliotheken an.
Im Gegensatz zu den CSPs, die eine komplette Web-Applikation mit User-Interface darstellen, ist die Entwicklung eines Web Service also nur eine weitere XML-Sicht auf das DBMS. Eine Anwendung, um auf die Methoden zuzugreifen, die als Web Services bereitgestellt werden, ist zusätzlich nötig.

5.2.2. Evaluation

Im Gegensatz zu der Entwicklung der Anwendung als Client/Server-Applikation sind die Web Services besser durch Caché unterstützt. Die Services können mit ein paar Mausklicks durch das Web Services Form Wizard oder durch Einfügen von [WebMethod] in die Methodendefinition leicht erstellt werden. Auf der Clientseite findet durch Caché allerdings keine Unterstützung statt. Zwar wird durch die Verwendung von SOAP die Entwicklung der clientseitigen Anwendung mit vielen Programmiersprachen ermöglicht, doch da die Mitarbeiter/innen des OSPs bisher größtenteils nur Erfahrung in der M-Programmierung haben, ist dies kein entscheidender Vorteil für die Web Services.
Positiv zu bewerten sind auch die offenen Standards, die verwendet werden. Dadurch fallen keine Lizenzkosten an und durch die dazugehörigen allgegenwärtigen, internetbasierten Technologien sind die Web Services vielerorts einsetzbar. Durch die Verwendung von HTTP zur Datenübertragung ist die Anwendung sehr kompatibel und es treten zudem nur selten Probleme mit der Firewall auf, die in Krankhäusern aufgrund der hohen Sicherheitsanforderungen streng eingerichtet ist. Durch die Flexibilität ist man aber nicht an HTTP gebunden, auch andere Protokolle können benutzt werden. Dies ist allerdings nicht zu empfehlen, da der Einsatz von HTTP und XML zu

einer offenen und flexiblen Architektur führt und die Anwendung dadurch unabhängig von Plattformen, Programmiersprachen und Protokollen ist.
Ein großer Nachteil der Verwendung von XML sind allerdings die Folgen für die Performance der Anwendung. Durch XML wird nicht nur die Flexibilität und die Verständlichkeit erhöht, sondern leider auch der Overhead. Die Performance spielt bei einer so komplexen Datenbank mit vielen Datensätzen, Tabellen und Merkmalen wie der Tumordatenbank aber eine entscheidende Rolle. Dieser Aspekt ist daher nicht zu vernachlässigen. Negativ auf die Entscheidung, welche Technologie bevorzugt werden soll, wirkt sich bei der Anwendungsentwicklung mittels Web Services auch das Thema Sicherheit aus. Die Web Services müssten verschlüsselt werden oder es müsste eine Authentifizierung stattfinden.
Da die Web Services nur eine weitere XML-Sicht auf das DBMS darstellen und keine komplette Web-Applikation mit User-Interface bieten, besteht auch hier wie bei der Umsetzung mittels Client/Server-Technologie die Schwierigkeit, dass eine Anwendung auf den Clients installiert werden müsste. Zwar beschränken sich die Änderungen während des Betriebs fast ausschließlich auf die serverseitigen Web Services, doch wäre trotzdem vor Inbetriebnahme der Anwendung eine Absprache zwischen den EDV-Abteilungen der Kliniken nötig, da der OSP keine Administratorrechte für die PCs der Dokumentarinnen und Dokumentare in den anderen Krankenhäusern hat.

- Darf etwas auf den Arbeitsplätzen der Dokumentarinnen und Dokumentare in den anderen Kliniken installiert werden?
- Wenn ja, wie viel und
- welche Voraussetzungen sind hard- und softwareseitig nötig, um die Anwendung überhaupt installieren und später benützen zu können?

Diese und viele weitere Fragen müssten ganz konkret beantwortet werden. Ohne das Ja auf die erste Frage, wäre die Anwendung zum Scheitern verurteilt, eine Umsetzung mittels Web Services nicht möglich. Nachdem alle Fragen geklärt sind, muss trotz allem noch der Arbeitsaufwand eingesetzt werden, die clientseitigen Anwendungen zu installieren.
Zudem wäre für die Entwicklung eine genaue Definition der Schnittstellen der Web Services nötig. Diese werden allerdings von Caché automatisch bereitgestellt.
Ein letzter Nachteil wäre der zusätzliche Web Server, der bei dieser Umsetzung benötigt würde. Zwar verfügt Caché über einen eigenen, doch dieser ist nur zum Testen geeignet.
Abschließend ist zu sagen, dass Web Services durch ihre Unabhängigkeit von Plattformen, Programmiersprachen und Protokollen und durch den Einsatz von SOAP zur Kommunikation und HTTP zur Datenübertragung flexibler sind

als andere Technologien, aber auch nicht so allumfassend. Die Performance ist schlechter als bei der Umsetzung mittels CSPs und auch der Sicherheitsaspekt ist nicht zu vernachlässigen. Daher ist diese Alternative nicht empfehlenswert und scheidet aus.

5.3. Web Anwendung mittels Caché Server Pages

5.3.1. Analyse und Vorstellung

Mit Hilfe der von InterSystems entwickelten Technologie der Caché Server Pages (CSP) [KIW][INT][CACHE] können mit einem Web Server und Caché dynamische Inhalte bereitgestellt werden. Diese Technologie ist speziell auf die Verwendung mit einer objektorientierten Datenbank zugeschnitten. Nach einer Einlernzeit können in kürzester Zeit Web-Anwendungen erstellt und diese auch leicht gewartet werden. Der für die Anwendung nötige Caché-Applikations-Server, mit dem die Vorteile Sicherheit und Performance einhergehen, kann zusammen mit dem Datenbankserver auf einem System installiert werden. Die Verbindung erfolgt über das TCP/IP-Protokoll, genauer über den Caché-Port „1972“. Der Applikations-Server verfügt über eine Laufzeitumgebung zur schnellen Ausführung von Code und einen eingebauten HTTP-Server, über den die CSP ausgeliefert werden können. Für die bekannten Web Server wie Apache oder Microsoft Information Server gibt es Erweiterungen, die den Einsatz von CSP mit diesen Servern ermöglichen. Beim Einsatz eines separaten HTTP-Servers wird die Anfrage von dort über den installierten CSP-Gateway an den Caché-Applikations-Server weitergeleitet Abb. 5.9.). Der Applikations-Server arbeitet dann als Backend-System. Dieses zweistufige Konzept ermöglicht es dem System, die Anwendung auf mehrere Server zu verteilen, der Anwendungs- und Datenbank-Server bleibt dabei im Hintergrund verborgen. Erst nach der Verteilung der Anfragen durch den CSP-Gateway wird die CSP-Anwendung ausgeführt.

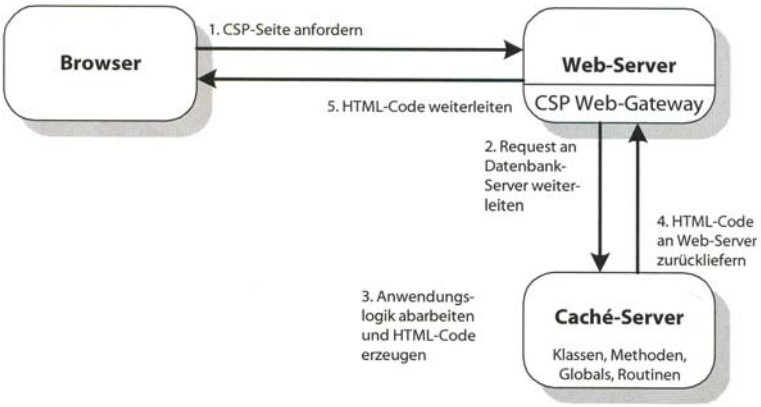

Abb. 5.9. Zusammenarbeit von Web- und Datenbank-Server bei der Verarbeitung von CSP-Seiten [KIW]

CSPs unterstützen sowohl HTML, wie auch XML und andere web-orientierte Mark-up-Sprachen. Daher können CSPs als HTML- oder XML-Dateien mit eingebetteten CSP-Tags, die beim ersten Aufruf von Caché automatisch in Klassendefinitionen mit ausführbarem Methodencode übersetzt und kompiliert werden oder direkt als Caché-Klassen entwickelt werden.

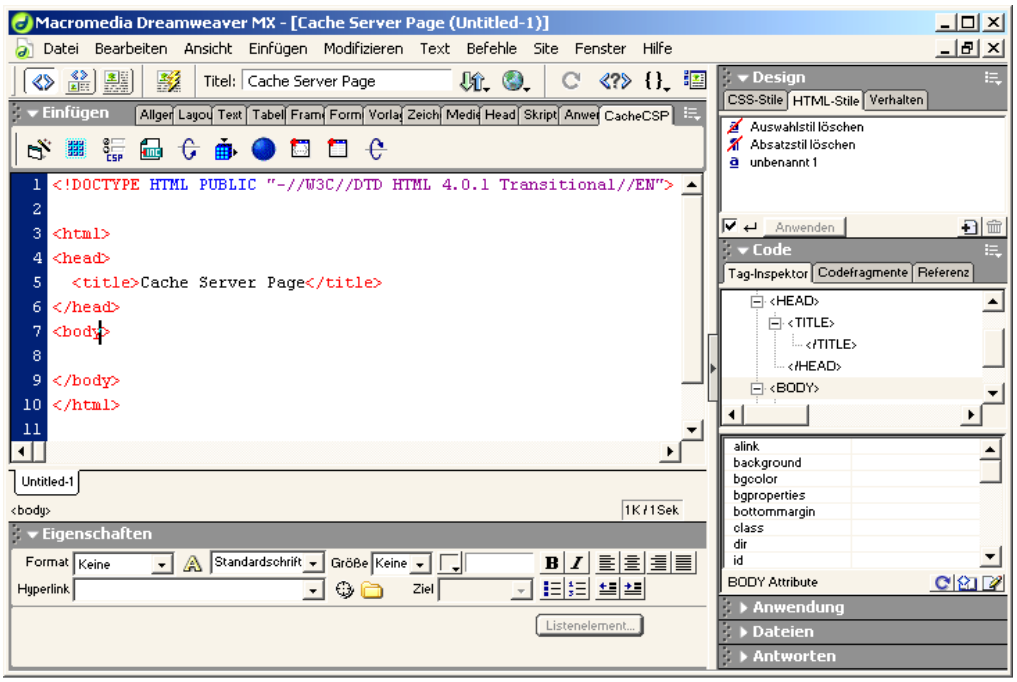

Abb. 5.10. Erstellen einer CSP mit dem Caché Studio

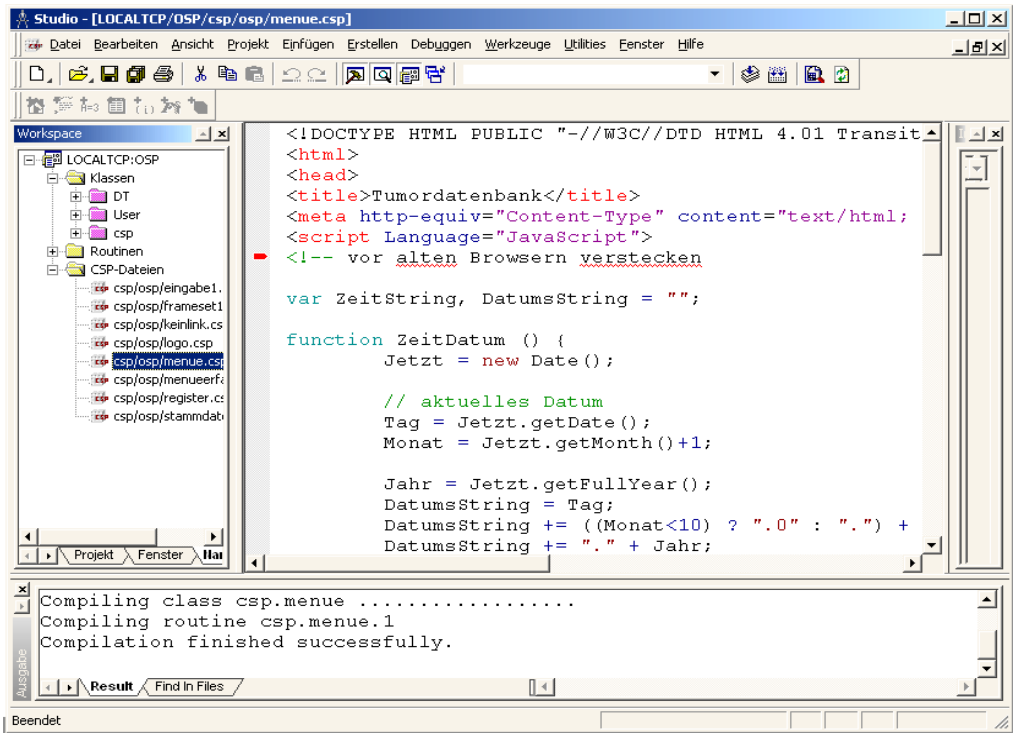

Abb. 5.11. Erstellen einer CSP mit Dreamweaver MX mit dem mitgelieferten Plug-in

Die CSP-Klassen können direkt im Caché-Studio (Abb. 5.10.), der Tag-basierte Ansatz in jedem HTML- oder Texteditor erstellt werden. Der OSP hat hierfür auf Anraten von InterSystems den Macromedia Dreamweaver MX (Abb. 5.11.) gekauft, der die CSP Technologie mit einem speziellen Plug-in unterstützt. Integrieren können die CSPs dabei HTML, XML, Bilder aber auch beliebig andere Binärdateien.

Die Caché Application Tags (CATs) sind wie HTML-Tags aufgebaut, formatieren aber nicht den Text, sondern führen Funktionen auf dem Datenserver und/oder im Browser aus. Eingefügt in eine HTML-Datei wird durch die CSP-Tags ein Anwendungscode für die Funktionen erzeugt. Zu den möglichen Funktionen, die mittels der Tags realisiert werden, gehören unter anderem Datenleseoperationen, um Felder in Formularen auszufüllen oder auszulesen. Die eingegebenen Daten können aber durch Tags auch validiert und in die Datenbank geschrieben werden. Durch IF- oder WHILE-Tags kann gesteuert werden, welcher HTML-Code im Browser angezeigt werden soll. Auch Hyper-Events können in CSPs eingebaut werden, um bei Browser-Events (Mausklicks, Timeouts...) Datenbankaktivitäten auszulösen. So können interaktive Web-Applikationen mit einer schnellen und flexiblen Reaktion auf Benutzereingaben erstellt werden, ohne den kompletten Formularinhalt an die Datenbank geschickt zu haben. Zusätzliche Flexibilität für den Programmierer wird auch durch die Möglichkeit erreicht, eigene Tags zu erstellen, die den jeweiligen Anforderungen der Anwendung genügen.

Die CSP-Tags befinden sich nur in der CSP-Datei, die an den Caché Web Compiler gesendet wird. Dieser wandelt sie automatisch in HTML-Standard um. So sind die CATs im HTML-Code nicht mehr eingebettet. (Abb. 5.12. und Abb. 5.13.). Der resultierende HTML-Code (Abb. 5.12.) wird vom Datenbank-Server erzeugt und an den anforderden Webserver geschickt. Dieser leitet den Code dann an den Browser weiter.

```
<select name="erfdatMonat" >
 <csp:loop counter=i from=0 to=12>
        <option>#(i)#</option>
 </csp:loop>
</select>
```

Abb. 5.12. WHILE-Tag in der CSP-Datei

```
<select name="erfdatMonat">
        <option>0</option>
        <option>1</option>
        <option>2</option>
        <option>3</option>
        <option>4</option>
        <option>5</option>
        <option>6</option>
        <option>7</option>
        <option>8</option>
        <option>9</option>
        <option>10</option>
        <option>11</option>
        <option>12</option>
</select>
```

Abb. 5.13. aus 5.12. automatisch erzeugter HTML-Code

Eines der wichtigsten CSP-Tags ist das „csp:QUERY“-Tag, mit dessen Hilfe SQL-Abfragen direkt in die HTML-Seite eingebaut werden können. Für die Vorbereitung und Ausführung der Datenbankabfrage sorgt dabei der Caché-Server zu Beginn der Seiten-Bearbeitung. Am Ende der Abfrage schließt er sie automatisch wieder. Der Tag arbeitet dabei wie folgt: er stellt die Verbindung zur ausgewählten Caché-Klasse her, führt die Abfrage aus und übergibt ein ResultSet an die CSP-Seite. Dieses ResultSet kann innerhalb der Seite verwendet werden. Um die in der Regel mehreren Datensätze des ResultSets auszugeben, wird das CSP:WHILE-Tag benutzt.

```
<CSP:WHILE CONDITION= (Queryname).Next()>
(Ausgabe der einzelnen Datensätze)
</CSP:WHILE>
```

Um einfach auf Objekte aus der Datenbank zuzugreifen, wird das „csp:OBJECT“-Tag benutzt. Dabei bekommt das Objekt beim Seitenaufruf eine eindeutige Objekt-ID. Diese wird über die Methode „%request.Get()“ bereitgestellt. Auch auf Variablen und Objekteigenschaften kann zugegriffen werden. Um diese auf der Seite darzustellen, wird „#(V*ariablenname oder Ausdruck*)#“ in den Code eingebaut .
Mit dem „csp:IF“-Tag und dem dazugehörigen „/csp:IF“ werden schließlich Seitenabschnitte eingeschlossen, deren Inhalte nur bei Erfüllung bestimmter Bedingungen generiert werden. Dank dieser Caché Application Tags sind Caché Server Pages mehr als nur Seitengestaltung. Die gesamte Anwendungslogik kann damit entwickelt werden.
In den HTML-Code können auch Caché-Scripts eingebunden werden.

```
<script language=cache runat=server>...</script>
```

Dieser in Caché ObjectScript geschriebene Code wird bei der Seitengenerierung zur Laufzeit ausgeführt. Auch eine Ausführung des Codes bei der Kompilierung ist möglich. Dazu wird „runat=server“ durch „runat=compile“ ersetzt. Mit dem Script-Tag können in die CSPs aber auch andere Scritptsprachen wie Javascript in die Seite integriert werden.
Jede CSP ist eine Unterklasse von %CSP.Page. Diese Systemklasse wird allerdings genauso wenig instanziert wie die einzelnen, von ihr abgeleiteten CSP-Klassen. Die gesamte Funktionalität ist in Klassenmethoden realisiert. Die Ausführung dieser Methoden ist ohne Instanzierung möglich. Eine Instanz ist dagegen für die drei Systemklassen %CSP.Request, %CSP.Session und für %CSP.Response nötig. Hierfür werden dem Programmierer Objektreferenzen von Caché automatisch zur Verfügung gestellt, um auf die jeweiligen Instanzen zuzugreifen. (Abb. 5.14.)

Klasse	OREF	Beschreibung
%CSP.Request	%request	Informationen über die aufgerufene HTTP-Methode einschließlich übergebener CGI-Variablen, Formularelemente und Query-Strings.
%CSP.Session	%session	Die Eigenschaften der aktuellen CSP-Sitzung einschließlich vom Programmierer zwischengespeicherter Werte.
%CSP.Response	%response	Eigenschaften, die die Antwort von Caché an den Web-Browser beeinflussen.

Abb. 5.14. %CSP-Klassen und ihre Objektreferenzen (OREF) [KIW]

Mit dem Session Model (%CSP.Session) wird die gesamte Verarbeitung für alle Seiten aus einem Browser als Teil einer Session aufgefasst. Die Session beginnt mit der ersten Browser-Anforderung und endet mit dem Schließen der Anwendung oder einem Timeout. Mit diesem Model wird das Problem gelöst, dass der Server normalerweise jede Anfrage vergisst, sobald sie beantwortet ist. Dieses Session-Management funktioniert für sichere Transaktionen mit HTTPS genauso gut wie für http. Dies ist für eine Anwendung im medizinischen Bereich mit den strikten Datenschutzregelungen auch nötig. Innerhalb einer Session kann eine Zwischenspeicherung der Anwendungsdaten auf dem Server erfolgen. Dies erleichtert die Entwicklung und Ausführung einer komplexen Anwendung, wie sie dieser Arbeit zugrunde liegt. Zwischen den einzelnen Browser-Anfragen werden die benötigten Daten automatisch und effizient erhalten und nicht als Teil des URL oder in versteckten Formularfeldern „aufbewahrt". Mit dem Session-Objekt von Caché können verschiedene Eigenschaften der Session kontrolliert werden. Das Objekt bleibt automatisch von einer Anfrage zur nächsten erhalten.
Neben dem Session-Objekt wird auch das Request-Objekt (%CSP.Request) von CSP bereitgestellt. Es beinhaltet alle Namen/Wert-Paare und Formulardaten mit anderen nützlichen Informationen.
In Caché ist es auch möglich, HTML-Formulare und ihre Felder direkt an ein Caché-Objekt und dessen Eigenschaften zu binden. Mittels CSPBIND ist dies der einfachste Umgang mit den Objekten für den Programmierer. So müssen keine zusätzlichen Methoden und Aktionen programmiert werden.

```
<CSP:OBJECT NAME=kerst CLASSNAME=User.KERST
        OBJECTID=#(%request.Data("KERSTID",1))#>
<FORM NAME KERST1 CSPBIND=kerst>
<INPUT TYPE=text NAME=klinr CSPBIND=klinr CSPREQUIRED>
<INPUT TYPE=text NAME=arzt CSPBIND=diagbem>
...
<INPUT TYPE=button VALUE="Speichern" Name=speichern1
        OnClick=KERST1_save();>
</FORM>
```

Ein Formularfeld mit dem Attribut CSPREQUIRED benötigt eine Eingabe, bevor es gespeichert werden kann. Dies wird über JavaScript-Elemente erreicht, die beim Kompilieren der CSP-Klassen erzeugt werden. Um die eingetragenen Daten in die Datenbank zu schreiben, wird die JavaScript-Methode *Formularname*_save() verwendet, die ebenfalls automatisch generiert wird.
Mit CSP kann die Anwendung in einem Modus betrieben werden, der für jeden Benutzer eine eigene Sitzung zwischen Web- und Datenbank- Server aufbaut. So werden die Anfragen eines Benutzers von einem Caché-Serverprozess bearbeitet, der eigens für ihn gestartet wird. Der Benutzer muss sich die Ressourcen also nicht mit anderen Benutzern teilen. Dieser State-Aware-Modus wirkt sich positiv auf die Performance und die Datensicherheit aus. Positiv für die Skalierbarkeit des Systems ist, dass der Webserver durch das Verarbeiten der Geschäftslogik nicht verlangsamt wird und mehr Browseranfragen bearbeiten kann, da die CSPs auf dem Datenserver ausgeführt werden und die Geschäftslogik und die Daten hierdurch eng verbunden sind.

5.3.2. Evaluation

Die Erstellung der Anwendung mittels Caché Server Pages stellte sich schnell als das Mittel der Wahl heraus. Diese Technologie wurde extra von InterSystems entwickelt, um schnell Webanwendungen zu erstellen und keine Installationen auf den Clients durchführen zu müssen. Der wichtigste Vorteil der Caché Server Pages ist, dass nichts auf dem Client installiert und upgedatet werden muss. Aufgrund der weiterhin zu erwartenden monatlichen Anpassungen oder Erweiterungen im Programm, verringert dies den Arbeitsaufwand erheblich. Auch die Absprachen mit den EDV-Abteilungen der OSP-Kliniken über Softwareinstallationen und gemeinsame Grundlagen bei der Hard- und Software, fallen dadurch weg. Es muss nur noch eine Abmachung bezüglich der Einstellung der Firewall getroffen werden.
Auch der Sicherheitsaspekt ist bei dieser Technologie gut gelöst. Zum einen ist ein Zugriff von außen strengstens zu unterbinden. Aufgrund der ärztlichen

Schweigepflicht und des Datenschutzes wäre dies sonst nicht nur grob fahrlässig, sondern sogar strafbar. Dies wird durch die Unterstützung von HTTPS und eine Authentifizierung der Benutzer mittels Login erreicht. Zum anderen ist auch eine Abschottung der Daten innerhalb des OSP unter den einzelnen Krankenhäusern notwendig. Aus Datenschutzgründen ist jedem Krankenhaus nur ein Lese- und Schreibzugriff auf die Daten der eigenen Patienten erlaubt. Ein Zugriff auf Daten anderer Patienten muss jedes Mal erst beantragt werden. Nicht nur die gezielte Freischaltung der Daten für die einzelnen Dokumentare und Dokumentarinnen ist hier wichtig, auch die Möglichkeit die Anwendung im State-Aware-Modus zu betreiben und somit jedem Benutzer seinen eigenen Prozess zuzuordnen, unterstützt die Einhaltung dieser Vorschrift. Hierdurch wird die Sicht auf die Anwendung der anderen Benutzer und damit deren Daten verhindert. Dieser Modus steigert aber auch die Performance, da sich die Benutzer die Ressourcen nicht mehr mit anderen Benutzern teilen müssen. Zum anderen wird die Programmierung der Anwendung vereinfacht, da der Programmkontext von einem zum nächsten Aufruf erhalten bleibt. Auch an die Weitergabe von Daten von einer Seite zur nächsten wurde beim Entwurf dieser Technologie gedacht. Durch das Session-Modell bleiben die für die Anwendung nötigen Daten zwischen den einzelnen Browser-Anfragen effizient erhalten.
Da bei der Entwicklung der Anwendung der Entwurf des Designs der Benutzeroberfläche von der Anwendungslogik getrennt werden kann, können für beide Aufgabenbereiche Spezialisten eingesetzt werden. Der Webdesigner entwickelt das Layout und der Programmierer baut in den HTML-Code des Designers die Funktionalität der Anwendung ein. So kann jeder in seiner gewohnten Entwicklungsumgebung arbeiten. Dies verringert die Einarbeitungszeit und damit auch die Entwicklungszeit. Viele bekannte Webtools können für die Gestaltung benutzt werden, für den Macromedia Dreamweaver MX gibt es sogar eine spezielles Plug-in, mit dem durch Point-and-Click Caché-Application Tags einfach eingebaut werden können.

5.4. Fazit

	Client/Server	Web Services	CSP
Entwicklungskosten	–	+/–	+
Lizenzkosten		Evtl. für Framework	+
Hardware	–	–	+
Performance	–	–	+
Sicherheit		–	+
Installation	–[1)]	–[1)]	+
Wartbarkeit	–[1)]	+/–	+
Bekanntheitsgrad	+	+	+/–
Erlernbarkeit	+/–	+/–	+/–
Support	Im Internet	Im Internet	Persönlich durch InterSystems

[1)] führt zur Abwertung

Abb. 5.15. Auswertungstabelle

In den vorangegangenen Kapiteln 5.1. bis 5.3. wurden die drei Möglichkeiten der Anwendungsentwicklung erklärt und erörtert. In der obigen Tabelle (Abb. 5.15.) sind die Vor- und Nachteile zusammengefasst.
Ausschlaggebend für die Entscheidung für die Entwicklung der Applikation mit Caché Server Pages waren die Probleme, die durch die nötige Installation von Teilen der Anwendung auf den Clients bei der Client/Server-Technologie und den Web Services entstehen würden. Autonome EDV-Abteilungen mehrerer Kliniken mit ihren ungleichen Organisatiosstrukturen, der unterschiedlichen Hardware und Software auf einen Nenner zu bringen und auf allen Arbeitsplätzen die gleichen Voraussetzungen für die Client-Applikation zu schaffen, stellt ein fast unüberwindbares Hindernis für die Verwendung derselben clientseitig installierten Anwendung dar. Negativ für die Client/Server-Technologie wirkte sich zusätzlich aus, dass alle Updates der Anwendung auf allen Clients zeitnah installiert werden müssten und dies organisatorisch kaum zu bewerkstelligen und vor allem mit einem riesigen Arbeitsaufwand verbunden wäre. Diese Argumente führten zum vorzeitigen Ausschluss der Client/Server-Technologie und auch der Web Services.
Bei den Caché Server Pages fällt die Installation auf den Clients weg, da die Anwendung komplett auf dem Caché-Server gespeichert ist. Die Clients müssen nur die aktiven Inhalte dieser dynamischen Seiten zulassen. Mehr ist clientseitig nicht erforderlich. Auch bei Änderungen in der Anwendung muss auf dem Client nichts getan werden. Die Änderungen können sogar im laufenden Betrieb eingebracht werden. Es muss nur darauf geachtet werden, dass zum Zeitpunkt des Austausches der alten Version der Anwendung kein

Prozess aus der alten Version eine andere Routine aufgerufen hat und danach zu der geänderten Routine zurückkehren möchte. Da für die Tumordatenbank kein 24 Stunden-Betrieb gewährleistet sein muss, kann dieses Problem umgangen werden, indem Updates eingespielt werden, wenn die Anwendung nicht genutzt wird.
Auch bei den Web Services können Änderungen an der Anwendung in das System eingebracht werden. Solange die Schnittstelle zum Client konsistent bleibt, kann der Web Service einfach verändert werden. Nur bei tiefgreifenden Änderungen, von denen auch die Schnittstellen betroffen wären, müssten die Anwendungen auf den Clients angepasst werden. Da dies die wenigsten Änderungen betrifft, fallen diese vor Ort in den einzelnen Kliniken notwendigen Updates nicht so ins Gewicht bei der Entwicklung mittels Web Services. Durch die Verwendung von XML erhöht sich allerdings der Overhead und damit verschlechtert sich die Performance der Anwendung. Dies wirkt sich bei solch riesigen Datenmengen wie der Tumordatenbank schnell aus und fördert daher nicht die Zufriedenheit der Benutzer mit der Anwendung. Außerdem würde die Erhöhung der Performance durch den Wechsel auf die Caché-Datenbank dadurch wieder zunichte gemacht werden. Bei den Caché Server Pages wird die Performance der Datenbank dagegen ausgenutzt, was die Anwendung schneller macht.
Die Entwicklungszeit wäre bei der Entwicklung einer Client/Server-Lösung am längsten, da sowohl die clientseitige als auch die serverseitige Anwendung selbst geschrieben werden müsste. Die Erstellung eines Web Service wird von Caché durch die Bereitstellung eines Form Wizards und eines Gateways zum Testen des Web Services erleichtert und damit die Entwicklungszeit dafür reduziert. Die Anwendung für den Client müsste hier allerdings von Hand geschrieben werden. Hier wird der Entwickler von Caché nicht unterstützt. Unterstützt und daher auch propagiert wird die Entwicklung von Caché Server Pages. Der Entwickler kann sich die Entwicklungsumgebung selbst aussuchen. Die Funktionalität und das Design können sogar getrennt entwickelt werden. Für den Macromedia Dreamweaver MX steht ein Plug-in zur Verfügung, aber die Entwicklung ist auch im Caché Studio und jedem Editor möglich. Aufgrund der schnellen Entwicklungsmöglichkeit des Point-and-Clicks im Dreamweaver und da keine Clientanwendung benötigt wird, reduziert sich die Entwicklungszeit hier erheblich. Die Entwicklungskosten sinken. Allerdings ist der Entwickler von CSPs beim Support auf InterSystems und seine Partner angewiesen. Aufgrund des geringeren Bekanntheitsgrads dieser Technologie ist im Internet kaum Hilfe dazu zu finden, ein paar wenige Foren gibt es, allerdings sind dort nur wenige Benutzer registriert. Bei auftretenden Problemen hilft InterSystems aber gerne weiter. Durch diesen persönlichen Support will die Firma die Zufriedenheit ihrer Kunden steigern und sie

langfristig an sich binden. Entscheidet sich der Entwickler für Client/Server oder Web Services, findet er im Internet viele Foren und Informationen. Die dort angeführten Beispiele müssen allerdings noch an Caché adaptiert werden. Die geringer Verbreitung von Caché in der gesamten IT-Welt und der damit verbundene niedrigere Bekanntheitsgrad der Caché Server Pages spielt bei der Akzeptanz der Anwendung allerdings keine Rolle, da Caché im medizinischen Bereich weit verbreitet und anerkannt ist.
Auch was die benötigte Hardware betrifft, sind Caché Server Pages im Vergleich zu den beiden anderen Lösungsansätzen zu bevorzugen. Es ist kein zusätzlicher Server nötig, der Platz und Administration benötigt. Für die CSP-Anwendung reicht der Caché-Server aus. Als Software und damit verbunden die dazugehörigen Lizenzkosten wurde vom OSP für die Entwicklung mittels CSPs nur der Macromedia Dreamweaver MX für 470€ angeschafft. Da die CSPs in jedem Editor erstellt werden können, wäre nicht einmal das erforderlich gewesen.
Bei der Erlernbarkeit der Entwicklungssprachen unterscheiden sich die drei Möglichkeiten wenig. Da die Mitarbeiter/Innen des OSP nicht aus dem Programmierumfeld kommen und nur Programmiererfahrungen mit M haben, sind alle Sprachen neu zu erlernen. Caché bietet zahlreiche Tutorials – jedoch bisher leider nur auf Englisch – und zahlreiche Beispiele in einer Beispieldatenbank, die an die eigenen Anforderungen angepasst werden können. Von InterSystems werden auch Fortbildungen und Kurse angeboten, die natürlich kostenpflichtig sind. Für die anderen Sprachen, die zur Entwicklung von Caché-Anwendungen verwendet werden können, finden sich im Internet zahlreiche Tutorials.
Der letzte Vorteil, der vor allem im Gesundheitswesen mit seinen strikten Datenschutzregeln besonders wichtig ist, bezieht sich auf den Sicherheitsaspekt. Es darf weder möglich sein von außen Daten auszulesen noch einzutragen noch darf innerhalb des Onkologischen Schwerpunktes auf alle Daten zugegriffen werden. Durch den State-Aware-Modus bei den Caché Server Pages werden die Anwendungen der Benutzer voneinander getrennt. Jedem Benutzer wird ein eigener Prozess zugeordnet. Bei den Web Services muss sichergestellt werden, dass entweder die Anwendung verschlüsselt ist und eine Authentifizierung stattfinden muss.
Abschließend ist zu sagen, dass die Web Services und die Client/Server-Architektur aufgrund der ausschlaggebenden Nachteile bei der Installation, Wartung und Performance für die Tumordatenbank am OSP mit seinen verteilten Krankhäusern mit unterschiedlichen Organisationsstrukturen nicht geeignet sind. Die Umsetzung der Anwendung mittels Caché Server Pages birgt keine schwerwiegenden Nachteile in sich und ist daher die Technologie, die am besten geeignet ist.

6. Durchführung

6.1. Vorgehensweise

6.1.1. Anforderungsanalyse

Zu Beginn der Studienarbeit stand die Einarbeitung in die Strukturen des Onkologischen Schwerpunktes Stuttgart und der Tumordatenbank. In dieser Phase standen folgende Fragen im Mittelpunkt der Arbeit:

- Wie ist der OSP aufgebaut und was sind seine Ziele?
- Von wem wird die Tumordatenbank genutzt?
- Wozu existiert diese Datenbank?
- Wie und wozu werden die Daten ausgewertet?
- Wie ist die Datenbank aufgebaut?
- Welchen Funktions- und Leistungsumfang hat sie?
- Wie funktioniert die zeichenbasierte Anwendung?
- Wie sieht der Aufbau der alten Datenbank MUMPS aus?
- Wie arbeitet der Datenbankgenerator KRAZTUR?

Nach Beantwortung dieser Fragen musste der Umgang mit der neuen Caché-Datenbank erlernt werden. Mit dem Caché Konfigurationsmanager kann man eine eigene Datenbank anlegen. Dabei wurde festgestellt, dass für manche Funktionen des Caché Cubes – ein Icon, das dem Benutzer die Möglichkeit bietet, Caché zu administrieren-, wie das Erstellen einer Datenbank, Administratorrechte nötig waren. Da diese aus Sicherheitsgründen nicht an jeden vergeben werden können, wurde die Testdatenbank über einen Administratoraccount erstellt. Im Caché Studio kann man der Datenbank dann neue Klassen hinzufügen.

Der nächste Schritt war die Analyse des Sollzustandes. Hier stand eine völlig neue Fragestellung im Mittelpunkt: welche Funktionen des alten Systems sollten beibehalten werden und welche Wünsche gab es für die neue graphische Oberfläche? Um die Funktionen, Anforderungen und Ergebnisse direkt mit den Anwendern abzusprechen und zurückzukoppeln, wurden mehrere Dokumentarinnen und Dokumentare mit Hilfe eines Prototyps in den Entwicklungsprozess einbezogen.

Eine der wichtigsten Anforderungen an die neue Datenbankanwendung ist natürlich die Benutzerfreundlichkeit. Mit Hilfe der Maus und der Tabulatortaste soll durch die ganze graphische Oberfläche navigiert werden können. Der Befehl für die Auswahl bei einem Global, wie zum Beispiel die Aufzählung der Krankenhäuser, soll durch Sprungmenüs überflüssig werden. Dadurch kann eine intuitive und weniger fehleranfällige Benutzerführung ermöglicht werden. Durch die Unabhängigkeit von der alten Oberfläche ist eine

Berücksichtigung der europäischen Richtlinie zu den ergonomischen Anforderungen für Bürotätigkeiten mit Bildschirmgeräten ISO 9241-10 möglich. Die weiße Schrift auf schwarzem Hintergrund kann durch eine ergonomisch sinnvollere schwarze Schrift auf weißem Grund ersetzt werden. Die Benutzerfreundlichkeit wird auch erhöht, da bei der graphischen Oberfläche viele Funktionen schneller erreicht werden können. Unnötige Zwischenschritte fallen durch den Einsatz von Navigationsleiste und Registerkarten weg. Auch Abkürzungsmöglichkeiten sind durch die Benutzung der Maus gegeben. So muss bei Änderungen nicht mehr der gesamte Dialog durchgegangen werden, sondern es kann direkt zu der entsprechenden Stelle gesprungen werden. Weitere Grundsätze der Dialoggestaltung wie die Selbstbeschreibungsfähigkeit und die Steuerbarkeit des Programms sind mit der neuen graphischen Oberfläche ebenfalls leichter umzusetzen. Durch die Möglichkeit, mit der Maus oder der Cursortaste auf der Anwendung zu navigieren, können Fehler bei der Dokumentation leichter und schneller behoben werden. Zusätzlich kann bei der Umstellung der Oberfläche auch die Chance genutzt werden, die Fehleranfälligkeit der Anwendung zu verringern. Durch neue Plausibilitätsprüfungen sollen Fehler ausgeschlossen werden. So soll es bei einer weiblichen Patientin zum Beispiel nicht möglich sein als Tumorart ein Prostata-Ca oder andere Tumoren an den männlichen Geschlechtsorganen zu dokumentieren. Vor dem Speichern sollen weitere Plausibilitätsprüfungen durchgeführt werden, wie zum Beispiel der Datumsabgleich. Eine Behandlung kann nicht vor dem eingetragenen Geburtsdatum oder vor der Diagnose durchgeführt werden. Auch dürfen bestimmte Felder nicht leer gelassen werden. Passiert dies doch, muss der Dokumentar/ die Dokumentarin darauf hingewiesen werden. Eine Speicherung ohne einen Wert für das Feld darf nicht erfolgen.

Weiterhin wünschenswert wäre, dass sich die Benutzer schnell in der Anwendung zurechtfinden. Dies ist durch die gleiche Reihenfolge bei den Dokumentationsbögen erreichbar.

Da die Zuständigkeit für die Benutzerarbeitsplätze der Dokumentare und Dokumentarinnen verschiedenen EDV-Abteilungen unterliegen, sind die Voraussetzungen auf der Hard- und auf der Softwareseite unterschiedlich. Sowohl bei der Verwirklichung mittels einer 2-Schichten Client/Server-Architektur als auch bei den drei-schichtigen Web Services ist eine Installation auf der Clientseite erforderlich. Hierfür wären zum einen genaue Absprachen zwischen den EDV-Abteilungen hinsichtlich der Menge und was überhaupt installiert werden darf notwendig. Zum anderen wären für die unterschiedlichen Betriebssysteme und die restliche installierte Software eventuell sogar verschiedene Anwendungen für die Clients nötig. Die Umsetzung einer Client/Server-Lösung oder eines Web Service mit Installation

einer Anwendung auf dem Client ist daher nicht realisierbar. Hiermit ist die Entscheidung für CSP gefallen.
Eine weitere wichtige Anforderung ist die Möglichkeit die Anwendung jederzeit leicht ändern und erweitern zu können. Da sich die Erkenntnisse und Anforderungen ständig erweitern und damit eine andere, genauere Dokumentation erforderlich machen, muss die Anwendung ständig angepasst werden. Dies muss ohne großen Aufwand möglich sein. Dieses Argument unterstreicht noch einmal die Entscheidung gegen die Client-Server-Lösung. Der Aufwand sicherzustellen, dass an jedem Benutzerarbeitsplatz ständig mit der aktuellsten an die Software des jeweiligen Arbeitsplatzes angepassten Version gearbeitet wird, ist zu groß.

6.1.2. Datenportierung

Anhand der Anforderungen aus Kapitel 6.1.1. konnte herausgearbeitet werden, welche Teile der alten Datenbank beibehalten werden können. Die Patientendaten zu übernehmen stellte kein großes Problem dar, bei der Anpassung der Routinen musste allerdings viel Handarbeit geleistet werden. Diese Steuerungsroutinen für die Benutzerführung sind noch notwendig, solange nicht alle Komponenten auf eine graphische Oberfläche umgestellt sind. Danach sind diese überflüssig. Die Benutzerführung der neuen graphischen Oberfläche ist Aufgabe des HTML-Codes.

6.1.3. Oberfläche

Nachdem das Anforderungsprofil der neuen Anwendung erstellt war, erfolgte die Planung der Oberfläche auf Papier. Nützlich für die graphische Oberfläche ist auf jeden Fall eine Navigationsleiste (Abb.6.1.). Auf dieser soll der Benutzer wie im jetzigen Hauptmenü zwischen Erfassungen, Überweisungen, Aufnahmekontrolle, Krankenakte, Auswertungen/ Listen, Sonderprogrammen und Hauptmenü navigieren können. Durch die Realisierung mit einer Navigationsleiste ist dies zu jedem Zeitpunkt in der Anwendung möglich. Ein weiterer Link, der über die Navigationsleiste direkt erreicht werden kann, ist der Logout. Weiterhin ist dort zu sehen, wer auf diesem Arbeitsplatz gerade eingeloggt ist und das Datum mit Uhrzeit. Da sich die Navigationsleiste im WWW auf der linken Seite etabliert hat, wird sie auch in der neuen graphischen Anwendung dort angebracht sein. Oben auf der rechten Seite wird der ausgewählte Patient angezeigt. Über Buttons soll ein neuer Patient dort angelegt oder der Patient gewechselt werden. Im Hauptfenster bleibt jetzt noch genug Platz für die Dokumentation.

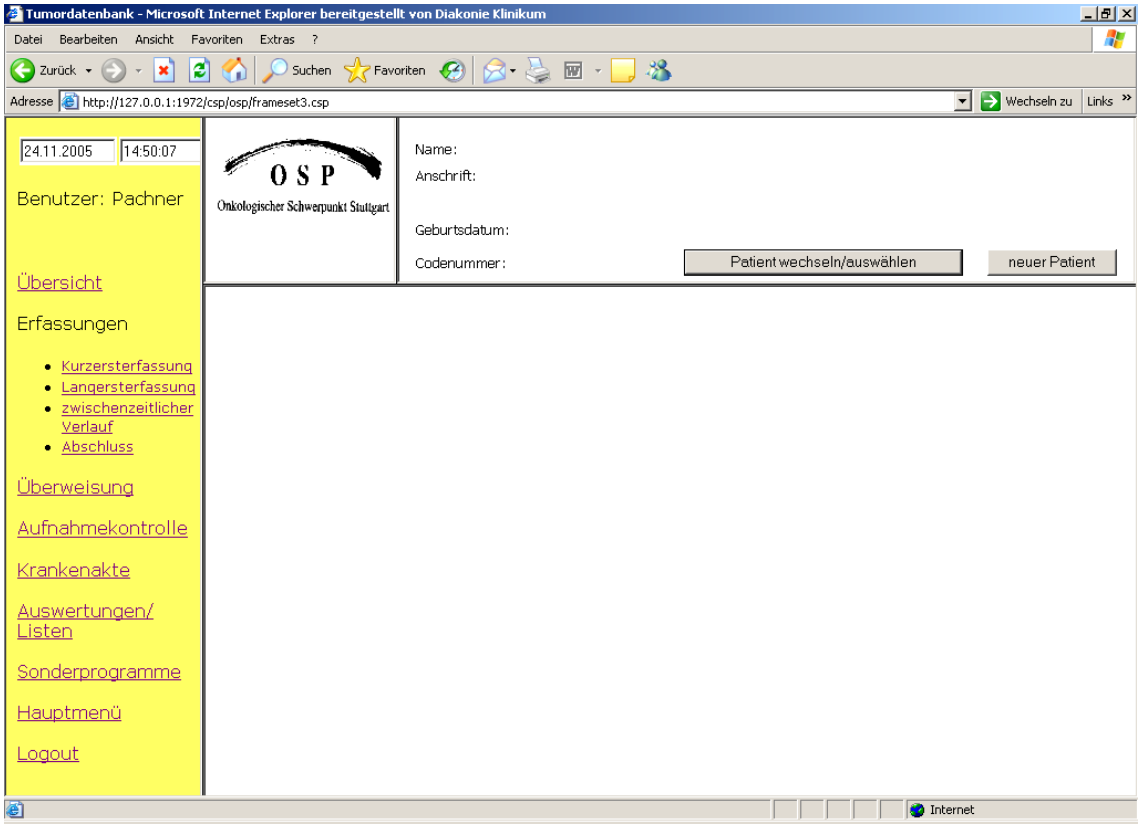

Abb.6.1. neue graphische Oberfläche mit Navigationsleiste, Patientendaten und Logo

Da ein Dokumentationsbogen im alten System aus mehreren Blöcken besteht und diese Struktur beibehalten werden soll, sind die einzelnen Blöcke über Register oder den Speicherbutton am Ende der Seite erreichbar. Dies fördert auch die Übersichtlichkeit der einzelnen Seiten, da die einzelnen Seiten nicht zu voll sind.

Mit Hilfe des Macromedia Dreamweavers und des Caché Studios wurde die Kurzersterfassung erstellt. Dieser Teil der Dokumentation ist übersichtlich, in sich abgeschlossen und daher durchschaubar. Anhand der KRAZTUR-Anwendung und der Datenbank wurde die neue Oberfläche angefertigt. Die Reihenfolge der Dokumentation wurde beibehalten, so dass die Einarbeitung für die Dokumentare und Dokumentarinnen nachher einfacher wird. Aus diesem Grund und weil es die Datenbank vorgibt, wurde auch die Struktur gleich gelassen. An den Stellen, an denen bis jetzt über zwei Fragezeichen angezeigt werden konnte, welche Auswahlmöglichkeiten vorhanden sind, wurden in der neuen Oberfläche Sprungmenüs eingesetzt. Die restlichen Felder sind Datums- und Freitextfelder. Ob diese Aufteilung bestehen bleibt, oder ob hinter manche Felder besser ein Global hinterlegt werden sollte, ist eine Angelegenheit der Datenbank und nicht Bestandteil dieser Arbeit. Wenn die Anwendung später jedoch komplett auf Caché umgestellt wird, ist zu

überlegen, ob nicht die Chance genutzt werden sollte, auch Änderungen in der Datenbankstruktur vorzunehmen.
Wichtig für eine schnelle Dokumentation ist, dass während der Arbeit mit dem Programm möglichst wenig zwischen Maus und Tastatur gewechselt werden muss. Die komplette Dokumentation eines Bogens sollte am besten nur mit der Tastatur möglich sein. Durch Realisierung des Designs mit Hilfe einer Tabelle springt der Curser mit der Tabulatortaste automatisch in das nächste Feld.
Nach Fertigstellung der Anwendung wurde diese 4 Dokumentarinnen und Dokumentaren präsentiert. Im Mittelpunkt dieser Treffen stand, ob alle Felder in die neue Anwendung integriert wurden, ob die Bedienung schnell genug und auch intuitiv ist und ob die Anordnung sinnvoll ist. Zu den wichtigsten Erkenntnissen, die aus diesen Treffen gezogen werden können, gehört, dass hinter die Sprungmenüs, in die nur Kürzel eingegeben werden, eine Ausgabe gehört, in der der Langtext nach Eingabe des Kürzels hineingeschrieben wird. Diese Ausgabe dient der Fehlervermeidung und Vergewisserung, dass auch das Richtige eingegeben wurde. Dies ist vor allem für Codierungen wichtig, die nicht so oft eingegeben werden. Das Datum wie im Prototyp als drei Sprungmenüs zu realisieren erwies sich als nicht so sinnvoll. Eine direkte Eingabe geht schneller. Es kam dazu der Vorschlag, dass das Jahr als Default gesetzt werden könnte, da bis auf das Geburtsjahr fast alle Daten vom aktuellen Jahr eingegeben werden. Manche Felder können weggelassen werden, da sie aus historischen Gründen entweder doppelt vorkamen (früher wurde erst auf Papier codiert und die Daten danach eventuell von einer anderen Person an einem anderen Tag in den Computer eingegeben) oder heute nicht mehr gebraucht werden.
In einigen Bereichen der Anwendung stellte sich allerdings heraus, dass Änderungen in der Datenbank sinnvoll wären, um die Anwendung konsistenter zu machen. Die heutigen unterschiedlichen Strukturen zwischen der Lang- und der Kurzersterhebung verwirren die Benutzer und führen zu unnötigen Fehlern, die durch Anpassung von Strukturen behoben werden könnten. Es wurde auch die Überlegung in den Raum gestellt, ob es nicht sinnvoll wäre, die Kurzersterhebung genauso wie später die Langersterhebung aufzubauen mit dem Unterschied, dass bei der Kurzersterhebung nicht alle Dokumentationsfelder angesprungen und alle Registerkarten ausgefüllt werden.
Um die Konsistenz in der endgültigen Anwendung sicherzustellen, muss daher zwischen den Dokumentaren abgesprochen werden, welche Änderungen wichtig sind und später bei der Dokumentation und Fehlersuche mehr Zeit einsparen als die Umstellung in der Datenbank an Arbeitsaufwand benötigt.

6.2. Erste Komponenten

6.2.1. Kurzersterfassung in KRAZTUR

kospv	**OSP-Verwaltungsdaten**	**Regeltest, Ferdinand, 01006666**	**>TES**
0	Tumor-ID-Nr.	2.	
1	Kliniknummer	Diakonissenkrankenhaus (Info. Von außerhalb)	
2	Erfassungsdatum	30.10.2003	
3.1.	Arzt		
5	Daten freigeben	ja	
	Korrektur ?	= ktum	

ktum	**Tumordaten**	**Regeltest, Ferdinand, 01006666**	**>TES**
0	Tumordiagnose, ICD10	Bösartige Neubildung des NierenbeckensC65	
1	Tumordiagnose, ICD9		
2	Tumordiagnose, Bemerkung		
3	Diagnosedatum	15.9.2001	
8	Tumorerkr., erste	fehlende Angaben	
9	Diagnosesicherung	histologisch gesichert	
4.1	Histo, ICD-0-DA	Invasives duktales Karzinom	
5.1	Histo, Grading		
6.1	Histo, Patho.-Nr.		
7.1.	Histo, Bemerkung		
4.2	Histo, ICD-0-DA		
	Korrektur ?	=kweitum	

kweitum	**Weitere Tumordaten**	**Regeltest, Ferdinand, 01006666**	**>TES**
1.1	Primärlok./Erstmanifestation	Nierenbecken	
2.1	Primärlokal., Bemerkungen		
1.2	Primärlok./Erstmanifestation		
19	Lokalisation, Seite		
3	Klassifikation	4.Aufl., 2.Revision	
4	TNM, T		
5	TNM, Tc		
6	TNM, N		
7	TNM, Nc		
8	TNM, M		
9	TNM, Mc		
13	patho-histologisch?	Ja (patho-histologisch)	
14	y-Symbol	nein	
10	LK, befallene (nur solide CA)		
11.1	Metastasen, init., Lokalisation		
	Korrektur ?	=kbge	

kbge	Bogeneingabe	Regeltest, Ferdinand, 01006666	>TES

1	Codierung	Regelmann
2	Eingabe	Regelmann
3	Eingabedatum	30.10.2003
4.1	Änderung	
6	Doku-Stand	15.9.2001
	Korrektur ?	

Abb. 6.2. Kurzersterfassung in KRAZTUR

In der Kurzersterfassung in KRAZTUR (Abb. 6.2.) stehen auf dem Bildschirm nur das nächste und die schon eingegebenen Eingabefelder. Je nach Eingabe entscheidet das Programm, welches Feld als nächstes erforderlich ist. Da z.B. in kbge keine Angaben bei Änderung eingetragen wurden, überspringt das Programm 5.1 Änderungsdatum. Wäre hier der Name der Person, die die Änderung durchgeführt hat, eingetragen worden und danach das Datum der Änderung, würde das Programm anschließend automatisch nach einer weiteren Änderung fragen (siehe zum Beispiel Primärlok./Erstmanifestation in kweitum). Diese Wiederholgruppen werden durchnummeriert und alle Daten gespeichert.
Aus manchen Eingaben folgt, dass andere Felder nicht ausgefüllt werden müssen oder können. Diese werden dem Benutzer in KRAZTUR dann nicht zur Verfügung gestellt. Andere Eingaben werden heute in der Kurzersterfassung nicht mehr dokumentiert (zum Beispiel die Angaben über Rezidiv, Procedere, Therapie oder allgemeine Bemerkungen siehe Abschnitt 6.2.1. Abb.6.7.), sind in alten Daten aber noch vorhanden. Zu diesen Daten kann der Benutzer durch Eingabe des Bogennamens bei Korrektur gelangen. Dort kann der Dokumentar oder die Dokumentarin Änderungen vornehmen oder Daten auslesen.
Ist ein Feld falsch eingegeben worden, muss der Benutzer zu dem falsch codierten Eingabefeld springen (: *Nummer der Zeile in KRAZTUR*) und seine Eingabe verbessern. Danach muss der Dokumentar oder die Dokumentarin überprüfen, ob die danach codierten Felder noch richtig sind und diese gegebenenfalls auch verbessern: Das Programm führt bei der Korrektur keine automatische Konsistenzprüfung durch.

6.2.2. Die neue graphische Oberfläche

In den Abbildungen 6.3. bis 6.7. ist die neue Dokumentationsoberfläche dargestellt. Diese wird jeweils ins Mainframe der Abb. 6.1. eingefügt. Zu erreichen ist die Kurzersterfassung über den Link Erfassungen in der Navigationsleiste und dort unter der Kategorie Kurzersterfassung. Durch den Button „speichern und weiter“ auf den Seiten 1-3 erfolgt eine automatische Weiterleitung auf die nächste Seite der Dokumentation. Die Seite 5 ist nur über die Registerkarte oben erreichbar, da in diese nichts mehr eingetragen wird. Hier können über den Link in der Registerkarte alte Daten aus der Datenbank ausgelesen und verändert werden. Die Seiten 1 bis 4 sind auch über die Registerkarten erreichbar, wenn gezielt nur auf einzelnen Bögen etwas geändert werden soll.
Zurzeit laufen noch der Test und die Evaluierung der Anwendung. Auch muss die Funktionalität der Anwendung noch erweitert werden und die Anbindung an das alte System fehlt.

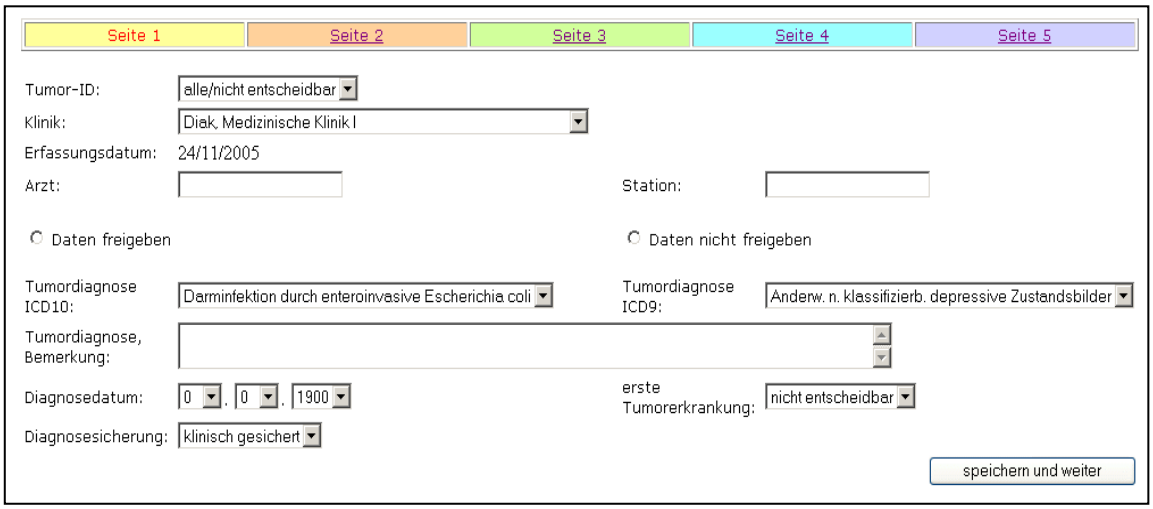

Abb. 6.3. Seite 1 der neuen graphischen Oberfläche

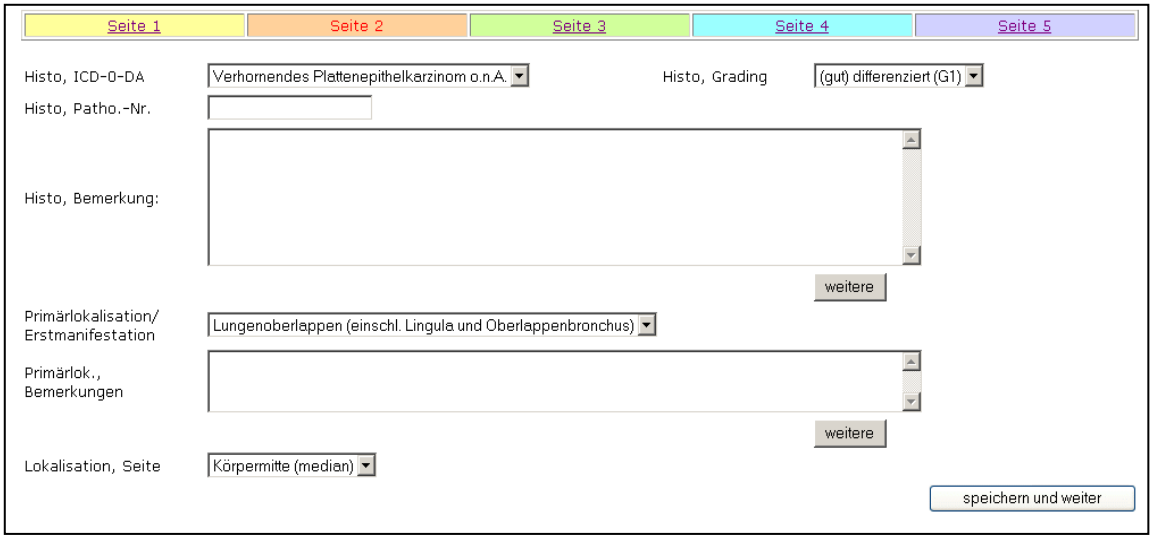

Abb. 6.4. Seite 2 der neuen graphischen Oberfläche

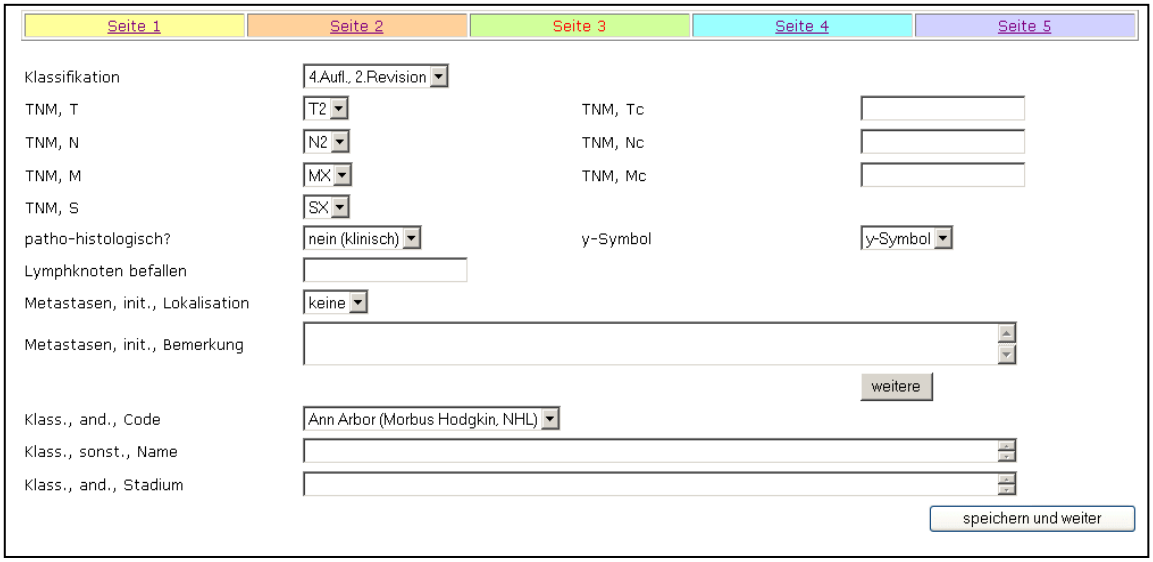

Abb. 6.5. Seite 3 der neuen graphischen Oberfläche

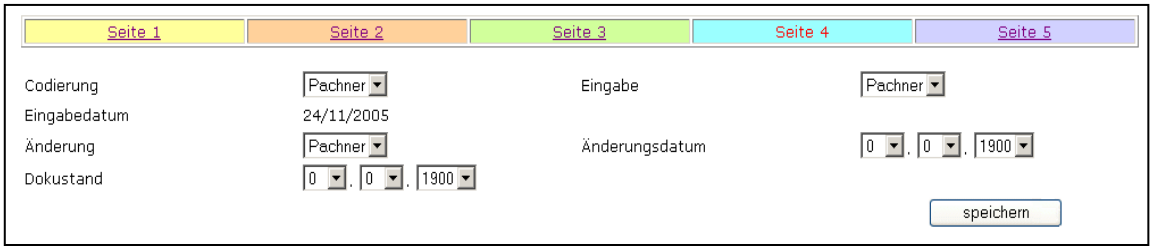

Abb. 6.6. Seite 4 der neuen graphischen Oberfläche

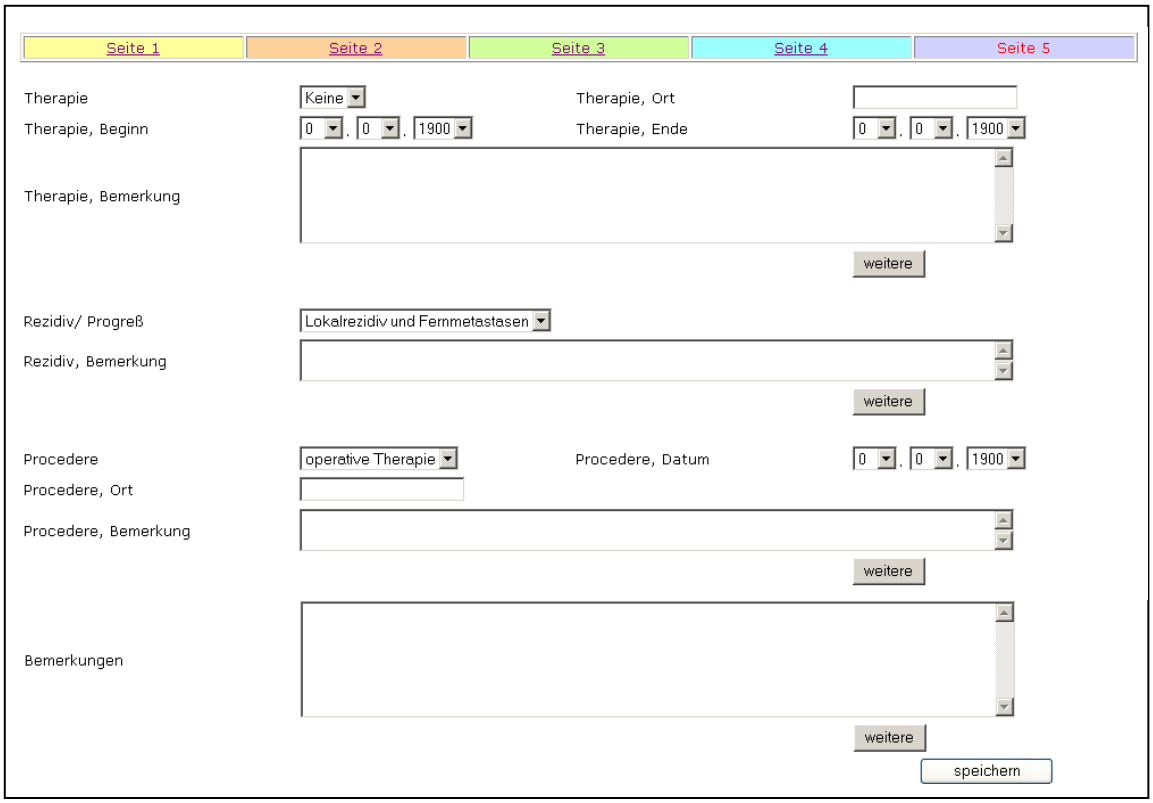

Abb. 6.6. Seite 5 der neuen graphischen Oberfläche

7. Zusammenfassung

Die am Onkologischen Schwerpunkt Stuttgart e.V. (OSP) eingesetzte Caché-Datenbank führt ein Insiderleben in der Medizininformatik als embedded System. Mit ihr ist es möglich, die Anwendung sowohl als Client/Server-Applikation, als auch auf Basis von Web Services oder als Webanwendung mittels Caché Server Pages (CSP) zu realisieren.
Die drei Möglichkeiten der Umsetzung unterscheiden sich hauptsächlich darin, wie viel auf dem Client installiert wird. Hierin liegt auch der Hauptvorteil der Caché Server Pages. Da bei dieser Variante keine zusätzlichen Programme oder DLL's auf dem Client installiert werden und die Firewall nur die entsprechenden Ports freischalten muss, ist diese Lösung den anderen vorzuziehen. Durch den Wegfall der Softwareinstallation auf dem Client werden viele Absprachen zwischen den verschiedenen EDV-Abteilungen der Kliniken, die für die Benutzerarbeitsplätze der Dokumentarinnen und Dokumentare zuständig sind, unnötig. Bei der Entwicklung der Anwendung müssen die Softwarevoraussetzungen auf den Clients nicht beachtet werden. Dadurch ist nur eine Version der Anwendung erforderlich. Zusätzlich sinken auch die Entwicklungskosten erheblich. Denn im Gegensatz zur Client/Server-Technologie und der clientseitigen Applikation bei den Web Services wird die Entwicklung von CSPs von Caché durch Wizards und Plug-ins unterstützt. Dies verkürzt die Entwicklungsdauer und die Einlernzeit merklich. Da das Design der Benutzeroberfläche von der Funktionalität getrennt entworfen werden kann, können für die beiden Bereiche Spezialisten eingesetzt werden, die jeweils in ihren bevorzugten Entwicklungsumgebungen arbeiten können. Auch dies kann die Entwicklungsdauer minimieren. Da die Caché Server Pages leicht geändert und die modifizierten Seiten leicht in die Anwendung eingefügt werden können, ist die Wartung des Programms unproblematisch. Durch die zentrale Speicherung der Anwendung auf dem Server ist sichergestellt, dass alle Dokumentarinnen und Dokumentare zu jedem Zeitpunkt mit derselben Version arbeiten.
Ein Nachteil von CSPs ist der fehlende Bekanntheitsgrad. Wenn man bei der Anwendungsentwicklung Hilfe benötigt, ist man auf InterSystems und die Partnerfirmen angewiesen. Im Internet gibt es kaum Foren zu diesem Thema. Mit Caché werden allerdings eine Reihe von Tutorials mitgeliefert, leider nur auf Englisch.
Die vielen Vorteile von Caché und die gravierenden Nachteile der beiden anderen Technologien gaben den Ausschlag für die Entscheidung für die Caché Server Pages.

Sowohl in Open M (früher MUMPS) als auch in Caché werden nicht nur die reinen Patientendaten, sondern unter anderem auch die Steuerroutinen für die Benutzerführung gespeichert. Die Patientendaten konnten in Caché ohne Probleme übernommen werden, die Routinen mussten allerdings angepasst werden. Solange für die Anwendung noch nicht die komplette graphische Oberfläche entwickelt ist, sind diese Routinen auch weiterhin nötig. Erst nach der vollständigen Umstellung der Tumordokumentation auf die graphische Oberfläche werden diese unnötig.

8. Literatur

[BVM] http://www.bvmi.de/hist_1969_03.htm, 24. November 2005

[CACHE] Diverse Unterlagen zu Cache, InterSystems Corporation, Cambridge

[DAT] http://www.datenschutz.hessen.de/tb22/k9p2.htm, 24. November 2005

[DER] DreamweaverMX, Einsteigerhandbuch, San Francisco, 2002

[ELE] http://www.elektronik-kompendium.de/sites/net/0503271.htm, 20. Oktober 2005

[ELK] Ellsässer, K.-D.; Trull, B.: Kraztur Systembeschreibung, 1988

[IDC1] Olofson, C.W.: InterSystems: Das bestgehütete Geheimnis der DBMS-Welt, IDC #33180, April 2005

[IDC2] Olofson, C.W.: Worldwide embedded Database Management Systems 2003 Vendor Shares, IDC # 32468, Dezember 2004

[IDM] Ide, M.: J2EE Programmier Handbuch, Einführung in die Java 2 Enterprise Edition, 2.Auflage Software & Support 2005

[INT] www.intersystems.de, Juni bis November 2005

[KIW] Kirsten, W.; Ihringer, M.; Kühn, M.; Röhrig, B.: Objektorientierte Anwendungsentwicklung mit der postrelationalen Datenbank Caché, 2.Auflage Springer 2003

[MUG] www.mug-d.de, 15. Oktober 2005

[NIK] Niemann, K.D.: Client/Server Architektur, vieweg, 1995

[OSP1] www.osp-stuttgart.de, Juni bis November 2005

[OSP2] OSP Jahresberichte 1988-2004

[OSP3] Handbuch für die OSP-Dokumentation, Juli 2005

[OSP4] diverse interne Arbeiten des OSP

[TEK] Terplan, K.: Client/Server Management, Datacom, 1995

[TIW] Tiehnen, W. von: Client/Server, 2.Auflage vieweg 1999

[WOE] Wolters, E.: Programmieren in MUMPS, Hanser Studienbücher, 1988

9. Anhang

9.1. Klassendefinition der Kurzersterfassung

9.1.1. Klasse KERST

```
/// Kurz-Ersterhebung
Class User.KERST Extends %Library.Persistent [ ClassType = persistent, Owner = _SYSTEM,
SqlRowIdName = IZU, SqlTableName = KERST, StorageStrategy = SQLStorage ]
{
Parameter EXTENTSIZE = 5000;
Relationship ChildKEbeharzt As User.KEbeharzt [ Cardinality = children, Inverse = IZU ];
Relationship ChildKEhisto As User.KEhisto [ Cardinality = children, Inverse = IZU ];
Relationship ChildKElok As User.KElok [ Cardinality = children, Inverse = IZU ];
Relationship ChildKEbehan As User.KEbehan [ Cardinality = children, Inverse = IZU ];
Relationship ChildKEmeta As User.KEmeta [ Cardinality = children, Inverse = IZU ];
Relationship ChildKEbem As User.KEbem [ Cardinality = children, Inverse = IZU ];
Relationship ChildKEaend As User.KEaend [ Cardinality = children, Inverse = IZU ];
/// Codenummer
Property code As %Library.String(COLLATION = "Exact", MAXLEN = 10, SELECTIVITY =
"0.0200%", TRUNCATE = 0) [ SqlColumnNumber = 2 ];
/// Aufnahmedatum
Property aufdat As DT.Datum(COLLATION = "Exact", MAXLEN = 10, SELECTIVITY =
"0.2000%", TRUNCATE = 0) [ SqlColumnNumber = 3 ];
/// Aufnahme-Nr.
Property aufnr As %Library.String(COLLATION = "Exact", MAXLEN = 20, SELECTIVITY = 1,
TRUNCATE = 0) [ SqlColumnNumber = 4 ];
/// Entlaßdatum
Property entdat As DT.Datum(COLLATION = "Exact", MAXLEN = 10, SELECTIVITY =
"0.2000%", TRUNCATE = 0) [ SqlColumnNumber = 5 ];
/// Sperrung durch Arzt
Property sperr As DT.IntExt(COLLATION = "Exact", GLOBAL = "OP", MAXLEN = 1,
SELECTIVITY = "33.3333%", SUB = "DS2", TRUNCATE = 0) [ SqlColumnNumber = 6 ];
/// Kliniknummer
Property klinr As DT.IntExt(COLLATION = "Exact", GLOBAL = "OP", MAXLEN = 4,
SELECTIVITY = "1.0000%", SUB = "KLINR1", TRUNCATE = 0) [ SqlColumnNumber = 7 ];
/// Erfassungsdatum
Property erfdat As DT.Datum(COLLATION = "Exact", MAXLEN = 10, SELECTIVITY =
"0.2000%", TRUNCATE = 0) [ SqlColumnNumber = 8 ];
/// Tumordiagnose, ICD9
Property diag As DT.IntExt(COLLATION = "Exact", GLOBAL = "OP", MAXLEN = 6,
SELECTIVITY = "0.1000%", SUB = "ND%Z", TRUNCATE = 0) [ SqlColumnNumber = 9 ];
/// Tumordiagnose, Bemerkung
Property diagbem As %Library.String(COLLATION = "Exact", MAXLEN = 50, SELECTIVITY =
1, TRUNCATE = 0) [ SqlColumnNumber = 10 ];
/// Diagnosedatum
Property diagdat As DT.Datum(COLLATION = "Exact", MAXLEN = 10, SELECTIVITY =
"0.2000%", TRUNCATE = 0) [ SqlColumnNumber = 11 ];
```

```
/// Klassifikation
Property klass As DT.IntExt(COLLATION = "Exact", GLOBAL = "OP", MAXLEN = 2,
SELECTIVITY = "14.2857%", SUB = "TUDA3", TRUNCATE = 0) [ SqlColumnNumber = 12 ];

/// T
Property T As DT.IntExt(COLLATION = "Exact", GLOBAL = "OP", MAXLEN = 5, SELECTIVITY
= "4.0000%", SUB = "OPJTUM4", TRUNCATE = 0) [ SqlColumnNumber = 13 ];

/// Tc
Property Tc As %Library.Numeric(MAXVAL = 4, MINVAL = 1, SCALE = 0, SELECTIVITY =
"20.0000%") [ SqlColumnNumber = 14 ];

/// N
Property N As DT.IntExt(COLLATION = "Exact", GLOBAL = "OP", MAXLEN = 6, SELECTIVITY
= "6.2500%", SUB = "OPJTUM6", TRUNCATE = 0) [ SqlColumnNumber = 15 ];

/// Nc
Property Nc As %Library.Numeric(SCALE = 0, SELECTIVITY = "20.0000%") [
SqlColumnNumber = 16 ];

/// M
Property M As DT.IntExt(COLLATION = "Exact", GLOBAL = "OP", MAXLEN = 6, SELECTIVITY
= "20.0000%", SUB = "OPJUM8", TRUNCATE = 0) [ SqlColumnNumber = 17 ];

/// Mc
Property Mc As %Library.Numeric(SCALE = 0, SELECTIVITY = "20.0000%") [
SqlColumnNumber = 18 ];

/// LK, befallene (nur solide Ca)
Property lkbefall As %Library.String(COLLATION = "Exact", MAXLEN = 6, SELECTIVITY =
"0.0600%", TRUNCATE = 0) [ SqlColumnNumber = 19 ];

/// y-Symbol
Property y As DT.IntExt(COLLATION = "Exact", GLOBAL = "OP", MAXLEN = 1, SELECTIVITY
= "16.6667%", SUB = "TUDA11", TRUNCATE = 0) [ SqlColumnNumber = 20 ];

/// Klass., and., Code
Property klasscode As %Library.String(COLLATION = "Exact", MAXLEN = 2, SELECTIVITY =
"7.6923%", TRUNCATE = 0) [ SqlColumnNumber = 21, SqlFieldName = klass_code ];

/// Klass., sonst., Name
Property klasssonst As %Library.String(COLLATION = "Exact", MAXLEN = 30, SELECTIVITY =
"0.2000%", TRUNCATE = 0) [ SqlColumnNumber = 22, SqlFieldName = klass_sonst ];

/// Klass., and., Stadium
Property klassstad As %Library.String(COLLATION = "Exact", MAXLEN = 30, SELECTIVITY =
"0.2000%", TRUNCATE = 0) [ SqlColumnNumber = 23, SqlFieldName = klass_stad ];

/// Rezidiv/Progreß
Property rez As DT.IntExt(COLLATION = "Exact", GLOBAL = "OP", MAXLEN = 1,
SELECTIVITY = "20.0000%", SUB = "LOK6", TRUNCATE = 0) [ SqlColumnNumber = 24 ];

/// Procedere
Property procart As %Library.String(COLLATION = "Exact", MAXLEN = 2, SELECTIVITY =
"6.6667%", TRUNCATE = 0) [ SqlColumnNumber = 25, SqlFieldName = proc_art ];

/// Procedere, Ort
Property procort As %Library.String(COLLATION = "Exact", MAXLEN = 50, SELECTIVITY =
"2.0000%", TRUNCATE = 0) [ SqlColumnNumber = 26, SqlFieldName = proc_ort ];

/// Procedere, Datum
Property procdat As %Library.String(COLLATION = "Exact", MAXLEN = 10, SELECTIVITY =
"2.0000%", TRUNCATE = 0) [ SqlColumnNumber = 27, SqlFieldName = proc_dat ];

/// Procedere, Bemerkung
Property procbem As %Library.String(COLLATION = "Exact", MAXLEN = 50, SELECTIVITY =
"2.0000%", TRUNCATE = 0) [ SqlColumnNumber = 28, SqlFieldName = proc_bem ];
```

```
/// Codierung
Property mdacode As %Library.String(COLLATION = "Exact", MAXLEN = 3, SELECTIVITY =
"2.5000%", TRUNCATE = 0) [ SqlColumnNumber = 29, SqlFieldName = mda_code ];

/// Eingabe
Property mdaein As %Library.String(COLLATION = "Exact", MAXLEN = 3, SELECTIVITY =
"2.5000%", TRUNCATE = 0) [ SqlColumnNumber = 30, SqlFieldName = mda_ein ];

/// Eingabedatum
Property eindat As DT.Datum(COLLATION = "Exact", MAXLEN = 10, SELECTIVITY =
"0.2000%", TRUNCATE = 0) [ SqlColumnNumber = 31 ];

/// Daten freigegeben
Property dafrei As DT.IntExt(COLLATION = "Exact", GLOBAL = "OP", MAXLEN = 1,
SELECTIVITY = "20.0000%", SUB = "DAFREI", TRUNCATE = 0) [ SqlColumnNumber = 32 ];

/// Rezidiv, Bemerkung
Property rezbem As %Library.String(COLLATION = "Exact", MAXLEN = 30, SELECTIVITY =
"1.0000%", TRUNCATE = 0) [ SqlColumnNumber = 33, SqlFieldName = rez_bem ];

/// Codierungszeit
Property codezeit As %Library.String(COLLATION = "Exact", MAXLEN = 1, SELECTIVITY =
"25.0000%", TRUNCATE = 0) [ SqlColumnNumber = 34, SqlFieldName = code_zeit ];

/// 3-stellige ICD
Property icd As DT.IntExt(COLLATION = "Exact", GLOBAL = "OP", MAXLEN = 3,
SELECTIVITY = "1.0000%", SUB = "ND%Z", TRUNCATE = 0) [ SqlColumnNumber = 35 ];

/// TNM
Property tnm As %Library.String(COLLATION = "Exact", MAXLEN = 25, SELECTIVITY =
"1.0000%", TRUNCATE = 0) [ SqlColumnNumber = 36, SqlComputeCode = { S
{tnm}=$$TNMKE^OSPSQLBO({IZU})
}, SqlComputed, Transient ];

/// Tumordiagnose, ICD10
Property diag10 As DT.IntExt(COLLATION = "Exact", GLOBAL = "ICD10", MAXLEN = 6,
SELECTIVITY = "0.0067%", SUB = "ICD10", TRUNCATE = 0) [ SqlColumnNumber = 37 ];

/// ICD10
Property icd10 As %Library.String(COLLATION = "Exact", MAXLEN = 3, SELECTIVITY =
"0.5000%", TRUNCATE = 0) [ SqlColumnNumber = 38, SqlFieldName = icd_10 ];

/// Diagnosesicherung
Property diagsich As DT.IntExt(COLLATION = "Exact", GLOBAL = "OP", MAXLEN = 1,
SELECTIVITY = "14.2857%", SUB = "WTD3", TRUNCATE = 0) [ SqlColumnNumber = 39 ];

/// Lokalisation, Seite
Property seite As DT.IntExt(COLLATION = "Exact", GLOBAL = "RU", MAXLEN = 2,
SELECTIVITY = "12.5000%", SUB = "TD6", TRUNCATE = 0) [ SqlColumnNumber = 40 ];

/// Tumor-ID-Nr.
Property tumorid As DT.IntExt(COLLATION = "Exact", GLOBAL = "OP", MAXLEN = 1,
SELECTIVITY = "14.2857%", SUB = "TUID", TRUNCATE = 0) [ SqlColumnNumber = 41 ];

/// Doku-Stand
Property stand As DT.Datum(COLLATION = "Exact", MAXLEN = 10, SELECTIVITY =
"0.2000%", TRUNCATE = 0) [ SqlColumnNumber = 42 ];

}
```

9.1.2. Klasse KEbeharzt

```
/// Behandelnder Arzt
Class User.KEbeharzt Extends %Library.Persistent [ ClassType = persistent, Owner =
_SYSTEM, SqlRowIdName = KE_beharzt_RowID, SqlTableName = KE_beharzt,
StorageStrategy = SQLStorage ]{
```

```
Parameter EXTENTSIZE = 5000;

Index RowIDBasedIDKeyIndex On VAR [ IdKey, PrimaryKey, Unique ];

/// Datensatznummer KERST
Relationship IZU As User.KERST [ Cardinality = parent, Inverse = ChildKEbeharzt, Required ];

/// Arzt
Property arzt As %Library.String(COLLATION = "Exact", MAXLEN = 30, SELECTIVITY = 1,
TRUNCATE = 0) [ SqlColumnNumber = 2 ];

/// Station
Property station As %Library.String(COLLATION = "Exact", MAXLEN = 10, SELECTIVITY = 1,
TRUNCATE = 0) [ SqlColumnNumber = 3 ];

/// Variablenname
Property VAR As %Library.String(COLLATION = "Exact", MAXLEN = 11, SELECTIVITY =
"20.0000%", TRUNCATE = 0) [ SqlColumnNumber = 4 ];

/// Zähler
Property Z As %Library.Numeric(SCALE = 0, SELECTIVITY = "20.0000%") [
SqlColumnNumber = 5, SqlComputeCode = { S {Z}=$P({VAR},"KBEARZT",2)
}, SqlComputed, Transient ];

}
```

9.1.3. Klasse KEhisto

```
/// Histologie
Class User.KEhisto Extends %Library.Persistent [ ClassType = persistent, Owner = _SYSTEM,
SqlRowIdName = KE_histo_RowID, SqlTableName = KE_histo, StorageStrategy = SQLStorage
]{

Parameter EXTENTSIZE = 5000;

Index RowIDBasedIDKeyIndex On VAR [ IdKey, PrimaryKey, Unique ];

/// Datensatznummer KERST
Relationship IZU As User.KERST [ Cardinality = parent, Inverse = ChildKEhisto, Required ];

/// Histo, ICD-O-DA
Property icdoda As %Library.String(COLLATION = "Exact", MAXLEN = 6, SELECTIVITY =
"0.0909%", TRUNCATE = 0) [ SqlColumnNumber = 2, SqlFieldName = icd_o_da ];

/// Histo, Grading
Property grading As %Library.String(COLLATION = "Exact", MAXLEN = 1, SELECTIVITY =
"14.2857%", TRUNCATE = 0) [ SqlColumnNumber = 3 ];

/// Histo, Patho.-Nr.
Property pathonr As %Library.String(COLLATION = "Exact", MAXLEN = 20, SELECTIVITY = 1,
TRUNCATE = 0) [ SqlColumnNumber = 4 ];

/// Histo, Bemerkung
Property bem As %Library.String(COLLATION = "Exact", MAXLEN = 50, SELECTIVITY = 1,
TRUNCATE = 0) [ SqlColumnNumber = 5 ];

/// Variablenname
Property VAR As %Library.String(COLLATION = "Exact", MAXLEN = 10, SELECTIVITY =
"20.0000%", TRUNCATE = 0) [ SqlColumnNumber = 6 ];

/// Zähler
Property Z As %Library.Numeric(SCALE = 0, SELECTIVITY = "20.0000%") [
SqlColumnNumber = 7, SqlComputeCode = { S {Z}=$P({VAR},"KHISTO",2)
}, SqlComputed, Transient ];
}
```

9.1.4. Klasse KElok

```
/// Primärtumor, Lokalisation
Class User.KElok Extends %Library.Persistent [ ClassType = persistent, Owner = _SYSTEM,
SqlRowIdName = KE_lok_RowID, SqlTableName = KE_lok, StorageStrategy = SQLStorage ]
{

Parameter EXTENTSIZE = 5000;

Index RowIDBasedIDKeyIndex On VAR [ IdKey, PrimaryKey, Unique ];

/// Datensatznummer KERST
Relationship IZU As User.KERST [ Cardinality = parent, Inverse = ChildKElok, Required ];

/// Primärlok./Erstmanifestation
Property lok As %Library.String(COLLATION = "Exact", MAXLEN = 6, SELECTIVITY =
"0.1429%", TRUNCATE = 0) [ SqlColumnNumber = 2 ];

/// Primärlokal., Bemerkungen
Property bem As %Library.String(COLLATION = "Exact", MAXLEN = 50, SELECTIVITY =
"0.1000%", TRUNCATE = 0) [ SqlColumnNumber = 3 ];

/// Variablenname
Property VAR As %Library.String(COLLATION = "Exact", MAXLEN = 9, SELECTIVITY =
"20.0000%", TRUNCATE = 0) [ SqlColumnNumber = 4 ];

/// Zähler
Property Z As %Library.Numeric(SCALE = 0, SELECTIVITY = "20.0000%") [
SqlColumnNumber = 5, SqlComputeCode = { S {Z}=$P({VAR},"KLOKP",2)
}, SqlComputed, Transient ];

}Klasse KEbehan
```

9.1.5. Klasse KEmeta

```
/// Initiale Metastasen
Class User.KEmeta Extends %Library.Persistent [ ClassType = persistent, Owner = _SYSTEM,
SqlRowIdName = KE_meta_RowID, SqlTableName = KE_meta, StorageStrategy =
SQLStorage ]
{

Parameter EXTENTSIZE = 100;

Index RowIDBasedIDKeyIndex On VAR [ IdKey, PrimaryKey, Unique ];

/// Datensatznummer KERST
Relationship IZU As User.KERST [ Cardinality = parent, Inverse = ChildKEmeta, Required ];

/// Metastasen, init., Lokalisation
Property lok As %Library.String(COLLATION = "Exact", MAXLEN = 2, SELECTIVITY =
"6.2500%", TRUNCATE = 0) [ SqlColumnNumber = 2 ];

/// Metastasen, init., Bem.
Property bem As %Library.String(COLLATION = "Exact", MAXLEN = 30, SELECTIVITY =
"20.0000%", TRUNCATE = 0) [ SqlColumnNumber = 3 ];

/// Variablenname
Property VAR As %Library.String(COLLATION = "Exact", MAXLEN = 10, SELECTIVITY =
"20.0000%", TRUNCATE = 0) [ SqlColumnNumber = 4 ];

/// Zähler
Property Z As %Library.Numeric(SCALE = 0, SELECTIVITY = "20.0000%") [
SqlColumnNumber = 5, SqlComputeCode = { S {Z}=$P({VAR},"KIMETA",2)
}, SqlComputed, Transient ];

}Klasse KEbem
```

9.1.6. Klasse KEaend

```
/// Bogenänderung
Class User.KEaend Extends %Library.Persistent [ ClassType = persistent, Owner = _SYSTEM,
SqlRowIdName = KE_aend_RowID, SqlTableName = KE_aend, StorageStrategy =
SQLStorage ]
{

Parameter EXTENTSIZE = 500;

Index RowIDBasedIDKeyIndex On VAR [ IdKey, PrimaryKey, Unique ];

/// Datensatznummer KERST
Relationship IZU As User.KERST [ Cardinality = parent, Inverse = ChildKEaend, Required ];

/// Änderung
Property mda As %Library.String(COLLATION = "Exact", MAXLEN = 3, SELECTIVITY =
"2.5000%", TRUNCATE = 0) [ SqlColumnNumber = 2 ];

/// Änderungsdatum
Property dat As %Library.String(COLLATION = "Exact", MAXLEN = 10, SELECTIVITY = 1,
TRUNCATE = 0) [ SqlColumnNumber = 3 ];

/// Variablenname
Property VAR As %Library.String(COLLATION = "Exact", MAXLEN = 9, SELECTIVITY =
"20.0000%", TRUNCATE = 0) [ SqlColumnNumber = 4 ];

/// Zähler
Property Z As %Library.Numeric(SCALE = 0, SELECTIVITY = "20.0000%") [
SqlColumnNumber = 5, SqlComputeCode = { S {Z}=$P({VAR},"KMDAG",2)
}, SqlComputed, Transient ];

}
```

Versicherung

Ich versichere, dass ich diese Studienarbeit selbständig gefertigt habe und die angegebenen Quellen und Hilfsmittel in einem vollständigen Verzeichnis enthalten sind. Außer den genannten wurden keine weiteren Hilfsmittel verwendet.

Stuttgart,_____ _____

(Andrea Pachner)