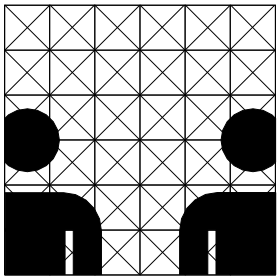

Universität Hamburg
Fachbereich Informatik
Verteilte Systeme und
Informationssysteme

Diplomarbeit

Digitale Identität und Identitäts-Management

Christian Philip Kunze

Matrikelnummer: 4923483
Emilienstraße 47
20259 Hamburg

Telefon: +49 40 | 430 977 56
E-Mail: 6kunze@informatik.uni-hamburg.de

Erstgutachter: Prof. Dr. W. Lamersdorf
Zweitgutachter: Prof. Dr. H. Züllighoven

Zusammenfassung

Wir alle bewegen uns heute wie selbstverständlich nicht nur in der realen, sondern auch in der virtuellen Welt. Dabei gehen wir im Alltag ganz natürlich mit unserer Identität und der unseres Gegenübers um. Nicht so in der Virtualität der weltumspannenden Netze. Hier herrscht eine vermeintliche Anonymität und bei vielen Tätigkeiten auch eine Isolation des Benutzers. Will man auch im virtuellen Raum mit zu mindest einem Teil seiner eigenen Persönlichkeit auftreten, so ist ein Konzept eines digitalen Äquivalents des zwischenmenschlichen Identitäts-Begriffs notwendig.

Diese Arbeit gibt einen Einblick in den vielschichtigen Begriff der Identität, so wie er alltäglich von uns gebraucht wird. Spricht man über die Identität, so muss man auch über den Umgang mit ihr sprechen. Dies führt zu dem Begriff des Identitäts-Managements.

Aus der Analyse der Begriffe der Identität und des Identitäts-Managements wird ein Kozept zur programmiersprachlichen Umsetzung entwickelt und deren prototypische Implementierung vorgestellt.

abstract

Today we all act naturally not only in the real but also in the virtual world. In doing so we handle native with our and foreign identities in every day life. But we do so not in the virtuality of interconnecting networks. In this situation often rules an assumed anonymity and an isolation of the user. If one likes to act with a part of his identity in virtual spaces, there is a need for a digital equivalent of the interpersonal comprehension of identity.

This thesis delivers insight to the complex term of identity as we use it in every day life. As discussing identity the need arises to argue how we present this identity in interaction. This again leads to the term of identity-management.

Based on the analysis of identity and identity management a programming concept is developed. Finally the implementation of a prototype is introduced.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Dieses Kapitel motiviert zunächst, warum eine Identitäts-Infrastruktur eine Bereicherung der heutigen Dienste und Anwendungen im Internet sein kann. Danach werden die Ziele dieser Arbeit vorgestellt sowie eine Eingrenzung des Themas vorgenommen. Abschließend wird das weitere Vorgehen dargelegt.

1.1 Motivation der Entwicklung einer Identitäts-Infrastruktur

Für immer mehr Menschen ist das Internet zu einem festen Bestandteil ihres Lebens geworden. Damit entwickelt es sich hin zu einem sozialen Raum, in dem die Nutzer einen Teil ihrer Arbeits- und Freizeit verbringen. Jedoch herrscht immer noch eine, wenn auch oftmals vermeintliche, Anonymität und Isolation des Nutzers vor. So bleibt denjenigen, die sich mit ihrem Browser auf Erkundung des World-Wide-Webs begeben, verborgen, wer sich mit ihnen durch das Netz bewegt. Die betrachteten Seiten scheinen nur für einen selbst bereitzustehen, dass eventuell gerade mehrere hundert oder tausend andere Surfer die selbe Seite betrachten, ist nicht zu erkennen. Nur wenige Anwendungen erlauben es zu mindest in einem begrenzten Rahmen diese Isolation aufzuheben. Hierzu zählen Programme wie ICQ oder der Messenger. Diese Systeme erlauben es Nutzern zu erkennen, ob sich ihre Freunde gerade ebenfalls im Internet bewegen und ihnen Nachrichten zukommen zu lassen. Eine Einbeziehung dieser Information in verfügbare Anwendungen außerhalb dieser Programme geschieht allerdings nicht. Auch die Bildung spontaner Gemeinschaften mit zuvor unbekannten Personen wird nicht unterstützt.

An dieser Stelle setzt die Idee der Etablierung und Nutzung einer Identitäts-Infrastruktur an. Der Nutzer soll sich als das erkennbar machen können, was er ist: ein Individuum mit einer eigenen Persönlichkeit. Er soll auch im Netz „jemand sein“. Die Dienste, die er nutzt, sollen ihn erkennen und ihrer Angebote auf ihn anpassen können. Durch Nutzung der Identitäts-Daten könnten ad hoc Gemeinschaften von Nutzern einer Webseite gebildet werden oder spontane Kooperationen entste-

hen. Das eindeutige Wiedererkennen von Netznutzern kann dabei die Basis für neue Vertrauens- und Reputationsmodelle sein.

Um dieses Szenario Wirklichkeit werden zu lassen, ist die Entwicklung eines möglichst allgemeinen und anwendungsunabhängigen Identitätsmodells notwendig. Aber auch ein Instrument zur Steuerung der Kommunikation und der übertragenen Inhalte –also ein Identitäts-Manager – muss integraler Bestandteil sein.

1.2 Vom Netznutzer zum Netzbürger

In diesem Abschnitt soll am Beispiel von Elli, einer begeisterten Schachspielerin, gezeigt werden, wie sich die Nutzung des Internets durch den Einsatz einer Identitäts-Infrastruktur verändert. Dazu wird ein Szenario skizziert und gezeigt, wie es zum einen mit heutiger Technologie und zum anderen mit einer Identitäts-Infrastruktur realisiert werden kann.

1.2.1 Szenariobeschreibung

Elli nutzt für ihr Hobby unterschiedliche Dienste und Angebote im Internet (siehe Abbildung 1.1). Sie hat sich bei einem Anbieter eines Schach-Servers angemeldet, über den sie mittels eines speziellen Programms andere Spieler treffen und gegen sie Partien spielen kann. Durch das Spielen gegen bewertete Gegner erhält Elli eine eigene Bewertungszahl, die ihre Spielstärke ausdrückt. Für die Dienste des Schachservers muss sie eine monatliche Gebühr entrichten, die von ihrer Kreditkarte abgebucht wird. Außerdem beteiligt sich Elli über ein Blackboardsystem[1] an Diskussionen über Schach. Sie kann eigene Beiträge einstellen, auf andere antworten oder Kommentare abgeben. Dieser Dienst steht allen Beteiligten kostenlos zur Verfügung. Da Elli unter anderem gern liest, kauft Sie bei verschiedenen Online-Shops Bücher und CDs über Schach. (vergl. [BZL03])

1.2.2 Das Szenario mit heutiger Technologie

Um den Schach-Server zu nutzen, muss sich Elli per E-Mail anmelden. Dazu überträgt sie Ihre persönlichen Daten und ihre Kreditkartennummer. Im Anschluss erhält sie vom Betreiber in einer Antwortmail ein Benutzernamen und ein Passwort zugewiesen. Mit diesen Daten muss sie sich gegenüber dem Server authentifizieren. Danach ist Elli in der Lage sich die benötigte Software herunterzuladen und Partien zu spielen. Die errechnete Bewertung wird ihrem auf dem Server gespeicherten persönlichen Datensatz hinzugefügt und ist innerhalb des Dienstes sichtbar. Auch für das Blackboardsystem muss sich Elli anmelden und erhält hier erneut Benutzername und Passwort. Für jeden Online-Shop, den Elli benutzt, muss sie erneut ihre persönlichen Daten und die Zahlungsinformationen eingeben. Daraufhin erhält sie wie bei den anderen Diensten ein

[1] Blackboardsysteme sind webbasierte Diskussionsforen

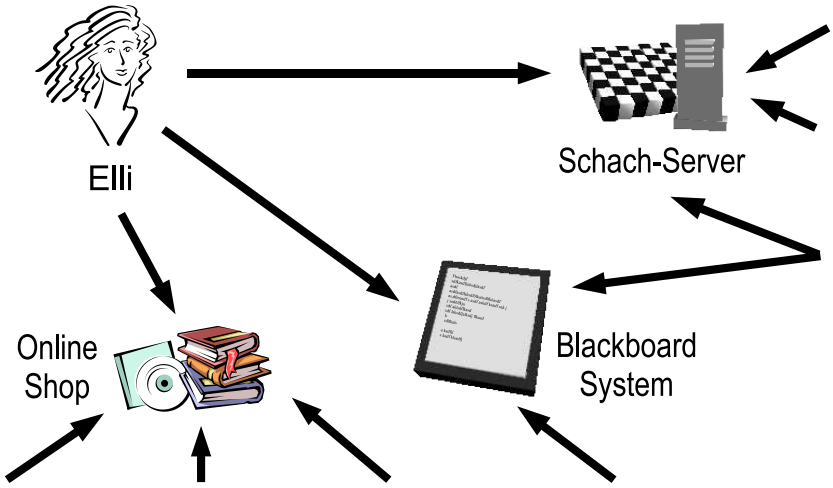

Abbildung 1.1: Szenario der Schachspielerin Elli

neues Passwort und einen neuen Benutzernamen. Für jeden Anbieter ist Elli zunächst ein unbeschriebenes Blatt, so dass von ihrer Vorliebe für Schach keiner weiß. Eine personenbezogene Präsentation der Angebote ist so nicht möglich. Dies ist, wenn überhaupt, erst nach Auswertung einer gewissen Anzahl von Transaktionen möglich. Diese Daten fallen jedoch für jeden Shop isoliert an. (nach [BZL03])

1.2.3 Das Szenario mit Unterstützung durch eine Identitäts-Infrastruktur

Anders sähe das Szenario aus, wenn Elli und die Anbieter eine Identitäts-Infrastruktur nutzen würden. Elli wäre in diesem Fall Besitzerin einer digitalen Identität, die Teile ihrer realen Identität auf die elektronische Welt abbildet. Diese enthält unter anderem Name, Adresse und die Kreditkarteninformation. Elli setzt sie bei allen Diensten zum Zweck der Identifizierung und Authentifizierung ein. Dabei können die Dienste – entsprechend ihrem Sicherheitsbedürfnis – verschiedene Authentifizierungs-Verfahren verwenden. Den Zugriff auf ihre Daten kann Elli durch die Vergabe oder Verweigerung von entsprechenden Rechten steuern. So dürfen zum Beispiel die Online-Shops und der Schach-Server die Kreditkartendaten abfragen, das Blackboardsystem hingegen nicht. Zusätzlich können die Anbieter auch Informationen in der digitalen Identität speichern, dies geschieht natürlich nur nachdem dies explizit erlaubt wurde. So werden zum Beispiel die Schachbewertung oder von Elli gekaufte Produkte Teil ihrer Identität. Um Manipulationen bei diesen Einträgen zu verhindern, könnten diese von den Einstellenden zusätzlich digital unterschrieben werden. Diese Daten stehen nun bei Bedarf anderen Diensten zur Verfügung. Dies führt dazu, dass Elli im Netz zunehmend mit ihrer Persönlichkeit auftritt und so als Individuum wahrgenommen wird. (nach [BZL03])

1.3 Ziel dieser Arbeit

Das Ziel dieser Arbeit ist es, auf der Basis einer Untersuchung des Begriffs der Identität eine möglichst allgemein einsetzbare Abbildung auf ein digitales Modell zu erarbeiten. Dabei soll das zwischenmenschliche Verständnis der Identität und deren Verwendung im Mittelpunkt stehen. Die Erkenntnisse bezüglich des Umgangs mit Identitäten sollen in einen Identitäts-Manager einfließen. Zusammen genommen, soll sich so eine Identitäts-Infrastruktur ergeben, die als Basis weiterer Anwendungen und Dienste dienen kann. Diese Infrastruktur soll in einem Prototypen realisiert werden.

1.4 Eingrenzung des Themas

Die vorliegende Arbeit ist innerhalb des Projektes onefC (open net environment for Citizens) entstanden. Sie soll Fragen der Sicherheit und des Vertrauens an den Stellen, wo es notwendig ist zwar berücksichtigen aber nicht konzeptionell ausformulieren oder implementieren. Dies gilt auch für die Gestaltung eines umfassenden Protokolls zum Austausch der Identitäts-Daten. Diese Bereiche sollen durch weitere und zum Teil bereits laufende Arbeiten abgedeckt werden. Aus diesem Grund sind einzelne Komponenten der entworfenen Infrastruktur lediglich durch ihre Schnittstellendefinition angegeben. Unter diesem Gesichtspunkt ist auch der vorgestellt Protokollansatz zu sehen.

1.5 Vorgehen

Das folgende Kapitel 2 befasst sich mit grundlegenden Technologien und Konzepten, die sich im Entwurf und der Realisierung der erarbeiteten Identitäts-Infrastruktur wiederfinden. Dies sind im einzelnen Konzepte zur global eindeutigen Bezeichnung und Referenzierung von Objekten, eine Übersicht über die abstrakte Datenbeschreibungssprache XML und die in der Arbeit verwendeten Entwurfsmuster. Zusätzlich werden die Java-Plattform und das zur Netzanbindung verwendete TCP/IP-Protokoll vorgestellt. Danach wird in Kapitel 3 der Begriff der Identität herausgearbeitet. Dazu wird das philosophische und psychologische Verständnis betrachtet. Zusätzlich werden seine technischen Aspekte bezüglich Informatik und der Gesellschaft aufgezeigt. Danach wird das Identitäts-Management und seine Aufgaben untersucht. Das Kapitel wird von der Beschreibung und dem Vergleich bestehender Ansätze abgeschlossen. Die so gewonnenen Erkenntnisse werden in Kapitel 4 in ein Konzept einer digitalen Identitäts-Infrastruktur umgesetzt. Dazu werden zunächst allgemeine Möglichkeiten des Aufbaus einer digitalen Identität betrachtet, aus denen dann das eigene Modell entwickelt wird. Danach wird der zur Architektur gehörende Identitäts-Manager entworfen. Abschließen wird ein Protokoll-Ansatz vorgestellt, mit dem der Prototyp getestet werden kann. Das Kapitel 5 geht auf ausgewählte Details der prototypischen Umsetzung der Identitäts-Infrastruktur ein. Anschließend werden in Kapitel 6

Szenarien vorgestellt, die typische Anwendungsfälle und ihre Lösungen mit der vorgestellten Identitäts-Infrastruktur aufzeigen. Die Arbeit wird von Kapitel 7 mit einer Zusammenfassung der Ergebnisse und einem Ausblick abgeschlossen.

Kapitel 2

Grundlegende Technologien und Konzepte

Dieses Kapitel soll Technologien und Konzepte vorstellen, die der in dieser Arbeit entwickelten Identitäts Infrastruktur zu Grunde liegen. Konkret werden Konzepte zur global eindeutigen Bezeichnung von Objekten, eine Möglichkeit zur abstrakten Datenrepräsentation, das Konzept der Java-Plattform und der Internet-Protokollstapel beschrieben. Zusätzlich werden noch Design-Muster vorgestellt, die beim Entwurf der konkreten Implementierung verwendet wurden.

2.1 Weltweit eindeutige Bezeichner

Weltweit eindeutige Bezeichner werden immer dann gebraucht, wenn man nicht lokal Referenzen erstellen möchte. Dieses Problem tritt immer wieder auf und hat schon verschiedenen Lösungen hervorgebracht. So zum Beispiel die beim Remote Procedure Call verwendeten UUIDs oder die im Internet üblichen URIs. Beide Bezeichnertypen können in der onefC-Architektur Einsatz finden und sollen hier vorgestellt werden.

2.1.1 Universal Unique Identifier

Universal Unique Identifier (UUID), auch bekannt als Globaly Unique Identifier (GUID), sind Bezeichner, die einzigartig sowohl im durch sie aufgestellten Raum als auch in der Zeit sind. Sie können für vielfältige Zwecke eingesetzt werden, wie zum Beispiel dem Markieren von kurz- oder langlebigen Objekten. Dabei haben UUIDs den großen Vorteil, dass zur Erzeugung der einzelnen Instanz keine zentrale Autorität benötigt wird. [OPE97]

Aufbau einer UUID

Ein Universal Unique Identifier besteht aus 128 Bits eingeteilt in 16 Oktette. Dabei haben die einzelnen Bytes die in Tabelle 2.1 angegebene Bedeutung, die folgend näher

erläutert wird:

Feldbezeichnung	Oktette	Beschreibung
time_low	0-3	Die unteren Bits des Zeitstempels.
time_mid	4-5	Die mittleren Bits des Zeitstempels.
time_hi_and_version	6-7	Die oberen Bits des Zeitstempels verbunden mit der Versionsnummer.
clock_seq_and_reserved	8	Die oberen Bits der Zeitsequenz verbunden mit der Variantennummer.
clock_seq_low	9	Die unteren Bits der Zeitsequenz.
node	10-15	Der räumlich eindeutige Bezeichner des Knotens.

Tabelle 2.1: Aufbau einer UUID nach [OPE97]

Zeitstempel Der in der UUID verwendete Zeitstempel, sollte in Universalzeit[1] angegeben werden. Ist dies nicht möglich, so kann auch die lokale Zeit verwendet werden, wobei jedoch auf eine einheitliche Verwendung geachtet werden soll. Der Zeitpunkt wird in Intervallen von 100-Nanosekunden ab 00:00:00.00 Uhr des 15. Oktober 1582 angegeben und ist 60 Bits lang. Dies Einschränkung der Länge führt dazu, dass das Zeitfeld ca. im Jahr 3400 überlaufen wird. Die fehlenden vier Bits des siebten Oktetts werden mit der Versionsnummer gefüllt [OPE97].

Version Das Versionsfeld besteht aus vier Bits, die als höchstwertige Bits an das letzte Byte des Zeitstempels angehängt werden. Die bisher vergebenen Nummern haben folgende Bedeutung:

1. [0001] Standardversion
2. [0010] DCE Sicherheitsversion mit eingebundenen POSIX[2] UIDs
3. [0011] Auf Namen basierende Version
4. [0100] Auf (Pseudo-)Zufallszahlen basierende Version

[1] Die Universalzeit oder Weltzeit ist die Zeit, die zum gewünschten lokalen Zeitpunkt in Greenwich (England) herrscht.

[2] Portable Operating System Interface der IEEE

Zeitsequenz Dieses Sequenzfeld ist eingefügt, um die Erzeugung von doppelten UUIDs durch Zurückstellen der Uhr oder Änderung der Knotenbezeichnung zu vermeiden. Ist die Zeit zurückgestellt worden, oder könnte dies geschehen sein, so wird der Wert der Zeitsequenz verändert. Ist der alte Wert bekannt, kann dieser inkrementiert werden. Ist er es nicht, so wird er zufällig neu gesetzt.

Wird die eindeutige Bezeichnung des Knotens geändert, z.B. in dem die Netzwerkkarte mit der eindeutigen MAC-Adresse getauscht wird, wird dieses Feld ebenfalls auf einen zufällig gewählten Wert gesetzt. Damit wird die Wahrscheinlichkeit der Erstellung doppelter UUIDs deutlich reduziert.

Variante Das Variantenfeld besteht aus 3 Bits, die als höchstwertige Bits an das 8. Oktett angehängt werden. Sie erlauben es die einzelnen Versionen näher zu spezifizieren. Die von DCE verwendeten Varianten sind als Beispiel in der Tabelle 2.2 aufgeführt.

MSB1	MSB2	MSB3	Beschreibung
0	–	–	Reserviert für die NCS Rückwärtskompatibilität
1	0	–	DCE Variante.
1	1	0	Reserviert für Microsofts GUID.
1	1	1	Reserviert für zukünftige Verwendung.

Tabelle 2.2: Das UUID Variantenfeld nach [OPE97]

Knoten Das Feld mit der Knotenbezeichnung soll so gewählt werden, dass eine räumliche Eindeutigkeit entsteht. Dazu wird im Allgemeinen die 48 Bit-stellige Adresse des Netzwerkanschlusses (MAC-Adresse) verwendet. Diese wird weltweit zentral vergeben und gewährleistet so die geforderte Eindeutigkeit.

Kann oder will man die MAC-Adresse nicht verwenden, so kann man die UUID Variante Vier verwenden, bei der die Adresse durch eine kryptografische Zufallzahl ersetzt wird. Um Kollisionen mit realen MAC-Adressen zu vermeiden, wird eine Besonderheit dieser Adressen ausgenutzt. Es existieren nämlich spezielle Multicast Adressen, die durch das Setzen eines speziellen Bits entstehen. Diese Rundsendeadressen werden nicht vergeben, so dass man durch setzen eben dieses Bits im generierten Knotenbezeichner eine Überschneidung vermeiden kann. [Gol99]

Repräsentation der UUID als Zeichenkette

Die Umsetzung einer UUID in eine standardisierte Zeichenkette erfolgt nach folgender Erweiterten Backus Nauer Form (nach [OPE97]):

UUID = <time_low> "-" <time_mid> "-"
<time_high_and_version> "-"
<clock_seq_and_reserved> <clock_seq_low> "-"
<node>

time_low = 4 * <hexOctet>

time_mid = 2 * <hexOctet>

time_high_and_version = 2 * <hexOctet>

clock_seq_and_reserved = <hexOctet>

clock_seq_low = <hexOctet>

node = 6 * <hexOctet>

hexOctet = 2 * <hexDigit>

hexDigit = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"|
"a"|"b"|"c"|"d"|"e"|"f"|"A"|"B"|"C"|"D"|"E"|"F"

Ein Beispiel einer UUID, die dieser EBNF genügt, kann folgendermaßen aussehen:

67ab7b3e-1f6f-4747-8c49-c829aac4bb76

Die Standardisierung der Abbildung einer UUID auf eine Zeichenkette ist besonders dort wichtig, wo diese zwischen verschiedenen Anwendungen ausgetauscht werden soll. Auf Grund der eindeutigen Semantik, kann dann eine solche Zeichenkette in jede beliebige lokale Repräsentationsform überführt werden.

2.1.2 Uniform Resource Identifier

Das Problem der eindeutigen Bezeichnung von Ressourcen stellt sich ebenfalls bei der Verwendung des Internets. Hier wurde zu dessen Lösung das Namensschema der Uniform Resource Identifier (URI) zum Referenzieren von Dokumenten eingeführt. [TS02] Die durch dieses Schema definierten Bezeichner bestehen aus Zeichenketten, mit denen auf einfache und erweiterbare Weise, abstrakte oder reale Objekte bzw. Ressourcen identifiziert werden können. [BL98]

Dabei wird im URI-Umfeld jegliches als Einheit identifiziertes Objekt als eine Ressource angesehen. Dies bedeutet, dass ein Uniform Resource Identifier nicht zwingend eine über ein Netzwerk ansprechbare Ressource identifiziert. Vielmehr können auch

körperliche Dinge wie Personen oder Bücher benannt werden. Der vergebene Bezeichner wird jeweils als stellvertretendes Objekt angesehen, welches eine Referenz zum eigentlichen Objekt herstellt. Dabei bildet der Bezeichner nicht auf eine fixe Ausprägung eines Objektes ab. Er stellt vielmehr einen Bezug zum gerade aktuellen Zustand eines Objektes her. [BL98]

Die Definition einer einheitlichen und gleichförmigen Syntax der URIs hat den Vorteil, dass, auch bei Verwendung verschiedener Bezeichnertypen innerhalb eines Kontextes, eine semantische Interpretation stets möglich ist. Auch die Einführung neuer Bezeichner und das Wiederverwenden alter URIs wird hierdurch vereinfacht. [BL98]

Das Design der Syntax der Uniform Resource Identifier basiert auf den Forderungen nach Erweiterbarkeit, Vollständigkeit und Fähigkeit zum Ausdrucken. Um neue Schemata dem URI-Namensraum hinzuzufügen, wurde der Bezeichner in die durch ein Doppelpunkt getrennten Teile der Schema-Bezeichnung und einer spezifische Zeichenkette aufgeteilt. Beim Umgang mit einer URI werden durch die Schema-Bezeichnung die Interpretations- und Verarbeitungsmethoden des spezifischen Teils festgelegt. Das Kriterium der Vollständigkeit fordert, dass für alle Namensschemata die Möglichkeit bestehen soll, sie durch Uniform Resource Identifier auszudrücken. Durch die Abstraktion der Bezeichner von der Ebene der Kodierung in Bits und Bytes auf die Ebene der Zeichen ist die Forderung nach Druckbarkeit gegeben. [BL94]

Grundlegende Strukturen aller Uniform Resource Identifier

Die im vorherigen Abschnitt angesprochene grundlegende Struktur der URIs ist die Einteilung in Schema-Bezeichnung und dem schemaspezifischen Teil:

<uri> = <schema> ":" <schemaspezifischer Teil>

Die Kodierung des Bezeichners wurde auf die Zeichen 'a' bis 'z' in Klein- und Großschreibung, die Ziffern '0' bis '9' sowie einige Sonderzeichen beschränkt. Dabei haben die Sonderzeichen jeweils eine festgelegte Bestimmung und können nicht wie normale Zeichen verwendet werden. Um Zeichen zu verwenden, die zwar im ASCII[3]-Zeichensatz enthalten sind, aber nicht direkt im Bezeichner enthalten sein dürfen, gibt es eine spezielle Kodierungsmethode. Diese wird als „Escaped Encoding" bezeichnet und durch ein Prozentzeichen gefolgt von der zweistelligen Hexadezimalzahl der Zeichenposition angegeben. Auf diese Weise lässt sich zum Beispiel ein Leerzeichen als „%20" darstellen und in URIs verwenden. [BL98]

Für die Bezeichnung von Ressourcen haben sich vor allem zwei Teilmengen der Uniform Resource Identifier durchgesetzt: zum einen die Uniform Resource Locator (URL) und zum anderen die Uniform Resource Name (URN). Dabei stellt eine URL eine ortsabhängige Referenz auf ein Dokument dar, die Informationen über die Art und

[3] American Standard Code for Information Interchange

Weise des Zugriffs auf die Daten enthält. URNs hingegen sind eindeutige Bezeichner, die ortsunabhängig langlebige Dokumente referenzieren. [TS02] Der Zusammenhang zwischen Uniform Resource Identifier, Locator und Name ist in Abbildung 2.1 verdeutlicht.

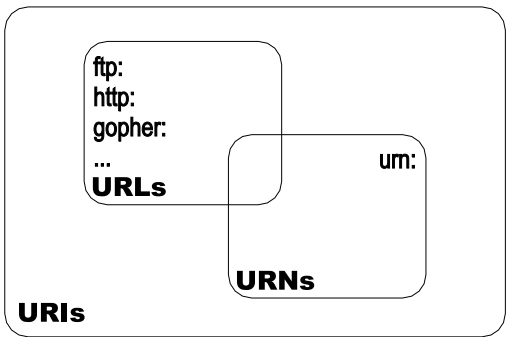

Abbildung 2.1: Zusammenhang zwischen URI, URL und URN nach [BL02]

Uniform Resource Locator

Uniform Resource Locators sind eine Teilmenge der URIs. Sie enthalten Angaben wo ein Dokument zu finden ist und wie auf es zugegriffen werden kann. Die Angabe der Zugriffsmethode geschieht dabei meist über das angegebene Schema. Die Ortsangabe ist im Allgemeinen durch einen DNS[4]-Eintrag und eine hierarchische Pfadangabe kodiert. [TS02] Diese Hierarchie wird durch das Sonderzeichen '/' als Trennsymbol der einzelnen Stufen kodiert. [BL94] Fehlt die explizite Angabe eines Dokumentes, so wird mit der URL ein Standarddokument bezeichnet.

Die Tabelle 2.3 zeigt einige Beispiele für Uniform Resource Locator. Dabei bezeichnet das Schema „http“ den Zugriff mittels des Hypertext Transfer Protocols oder „ftp“ mittels des File Transfer Protocols.

Da die URLs den Ort der Datenhaltung beinhalten, gibt es bei ihrer Verwendung das Problem, dass sich der Bezeichner beim Verschieben des Dokuments innerhalb oder zwischen Servern ändert. Dies führt bei Verwendung eines nicht mehr aktuellen Bezeichners zu sogenannten „gebrochenen Links“. Also Verweise, die ins Leere führen. [CDK01] Deshalb kann man sagen, dass URLs keine persistenten Bezeichner eines Dokuments sind. Sie sind nur so lange gültig, wie sich das Dokument an der in ihnen kodierten Stelle befindet. Die im onefC Kontext geforderte Eindeutigkeit besitzen sie allerdings.

Uniform Resource Name

Die URNs sind eingeführt worden, um das Problem der „gebrochenen Links“ bei der Verwendung von URLs zu lösen. URNs referenzieren im Gegensatz zu URLs Doku-

[4]Domain Name Service

Schema	Zugriffsmethode	Beispiel
http	HTTP	http://www.cs.vu.nl:80/globe
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README.txt
file	lokale Datei	file:/edu/book/work/chp/11/11
data	Inline Daten	data:tex/plain;charset=iso-8859-7,%e1%e2%e3
telnet	Remote Login	telnet://flits.cs.vu.nl
tel	Telefon	tel:+31201234567
modem	Modem	modem:+31201234567;type=v32/globe

Tabelle 2.3: URL Beispiele nach [TS02]

mente unabhängig vom Ort der Speicherung. Deshalb kann, nach einer Auflösung der URN in den jeweils aktuelle Speicherort, stets auf das Dokument zugegriffen werden. [CDK01]

Die allgemeine Struktur einer URN besteht aus dem URN-Schema mit der Bezeichnung „urn“, einem Bezeichner des Namensraumes und einem namensraumspezifischen Teil:

<URN> = “urn :“ <Namensraum> “:“ <namensraumspezifischer Teil>

Die Interpretation des spezifischen Teils hängt bei Uniform Resource Namen nicht vom Schema ab, sondern wird durch den Namensraum-Bezeichner definiert. Um Verwirrungen mit dem Schema zu vermeiden ist „urn“ als Bezeichner des Namensraumes nicht erlaubt. Mit einer URN wird zwar ein Dokument eindeutig referenziert, es kann jedoch nicht direkt der Ort der Speicherung oder die Zugriffsmethode ermittelt werden. Deshalb muss durch den Namensraum immer auch eine Methode zum Zugriff definiert werden. Dies kann zum Beispiel durch eine Abbildung auf eine URL geschehen. [Moa97]

Beispiele für Uniform Resource Namen sind die über die ISBN[5] identifizierenden Bücher-URNs:

urn:isbn:0-13-349945-6

Oder URNs zum identifizieren der RFCs[6] der Internet Engineering Task Force:

urn:ietf:rfc:2141

[5]Internationale Standard-Buchnummer
[6]Request For Comments

2.2 eXtensible Markup Language

Daten zwischen unterschiedlichen oder verschiedenen implementierten Systemen auszutauschen, ist meist nur dann möglich, wenn diese in einer programmiersprachlich unabhängigen Form vorliegen. Dies liegt darin begründet, dass die Strukturen und Repräsentationen der Datentypen der einzelnen Sprachen meist nicht kompatibel sind. Es muss also eine Beschreibung gefunden werden, die beide verstehen und die in die jeweilige lokale Struktur umgewandelt werden kann. Eine dieser abstrakten Repräsentationsformen für Daten ist die eXtensible Markup Language (XML). Sie wird von der onefC-Infrastruktur zum Austausch der Identitäts-Daten und deren persistente Speicherung verwendet. Aus diesem Grund soll XML an dieser Stelle vorgestellt werden.

Die eXtensible Markup Language ist eine Teilmenge der SGML[7]. Dabei ist bei ihrer Definition darauf geachtet worden, dass sie eine große Bandbreite von Anwendungen unterstützt, sie leicht von Programmen verarbeitet werden kann und möglichst wenige optionale Merkmale besitzt. Darüber hinaus sollte der Entwurf formal präzise und für Menschen lesbar sein.[Min02] Aus diesen Zielen, hat sich im Laufe der Zeit neben der allgemeinen Dokumentstruktur und der den XML-Einheiten zu Grunde liegenden Datentyp-Definitionen (DTD) eine Reihe von weiteren XML-Technologien entwickelt. Dies sind zum Beispiel die Transformation von XML-Dokumenten mittels XSLT [8], die Definition von Namensräumen oder die Adressierung von Dokumentteilen mit XPath. Die im Moment in der onefC-Architektur verwendeten Elemente der XML-Familie sind die XML-Dokumente als Träger der Identitäts-Daten und ihre Strukturdefinition durch XML-Schema, welches zwischenzeitlich die DTD abgelöst hat.

2.2.1 XML-Dokumente

XML-Dokumente sind textuelle Dokumente, basieren also auf Zeichen und nicht auf einer binären Darstellungsform. Damit ein Text als XML-Dokument gilt, muss er jedoch zusätzliche Eigenschaften erfüllen. Tut er dieses, so spricht man von einem wohlgeformten XML-Dokument.

Grundlegend besteht ein Dokument der eXtensible Markup Language aus einem Prolog, einem Element und optional beliebig vielen sonstigen Teilen, in genau dieser Reihenfolge. [Min02] Dabei sind die sonstigen Teile nicht näher spezifiziert. Sie haben jedoch die eine Einschränkung, dass sie keine XML-Elemente enthalten dürfen.

Prolog eines XML-Dokuments

Der XML-Prolog sollte stets mit der XML-Deklaration beginnen, um die verwendete Version der eXtensible Markup Language zu spezifizieren. Optional können in dieser

[7] Standard Generalized Markup Language
[8] Extensible Stylesheet Language Transformation

Deklaration weitere Angaben zur Kodierung oder Abhängigkeit von anderen Dokumenten angegeben werden. Die aktuelle Version ist die „1.0“. Ob es weitere Versionen geben wird, ist nicht ausgeschlossen aber im Moment wird nicht daran gearbeitet. Minimal sieht der Prolog folgendermaßen aus:

```
<?xml version="1.0"?>
```

Als weiteres Element des Prologs kann eine Dokumenttyp-Deklaration angegeben werden, die aus Markup-Deklarationen oder einem Verweis auf solche besteht. Mit deren Hilfe wird die Grammatik der verwendeten Dokument-Klasse beschrieben. [Min02] Durch diese Angabe kann neben der Wohlgeformtheit auch die Gültigkeit des Dokuments bestimmt werden. Diese ist genau dann gegeben, wenn das enthaltene XML-Element den Produktionen der Grammatik entspricht.

Das XML-Element

XML-Elemente sind hierarchisch angeordnet. In jedem Dokument darf es immer nur ein Element auf oberster Ebene geben. Dieses Wurzel- oder Dokument-Element ist niemals Teil eines anderen Elements. Außerdem gilt, dass Elemente nicht in einander verschachtelt sein dürfen. Das heißt, der Beginn und das Ende müssen jeweils in ein und dem selben Element sein. [Min02] Dabei werden der Start und das Ende durch sogenannte Tags gekennzeichnet. Diese werden durch spitze Klammern begrenzt und tragen jeweils den Namen des zu ihnen gehörenden Elements. Ein öffnendes Tag kann zusätzlich noch Attribute enthalten, die als Name/Wert-Paare angegeben werden. Um die Grenzen eines Wertes bestimmen zu können wird dieser wiederum durch Anführungszeichen begrenzt. Ein schließendes Tag wird durch die Angabe eines Schrägstrichs direkt nach der öffnenden Klammer definiert. Zwischen Anfang- und Endmarkierung wird dann der Inhalt des Elements angegeben. Dies können beliebige Zeichenketten und/oder weitere Elemente sein. Zur abkürzenden Schreibweise von leeren Elementen kann ein öffnendes Tag durch Angabe des Schrägstrichs vor der schließenden Klammer gleichzeitig zur Endmarkierung werden. Ein Element zur Angabe einer E-Mail-Adresse könnte als XML-Element folgendermaßen aussehen:

```
<email type="business">6kunze@informatik.uni-hamburg.de</email>
```

Die Angabe der Grammatik, mit der eine Klasse von Dokumenten beschrieben werden kann, wurde ursprünglich durch eine DTD angegeben. Diese bietet jedoch nur eingeschränkte Möglichkeiten der Strukturierung und Typisierung der Elemente eines XML-Dokuments, so dass ein neuer Mechanismus eingeführt wurde. Dieser ist das XML-Schema, mit dem auch die Dokument-Klassen der Identitäts-Daten und des Protokollansatz der vorliegenden Infrastruktur beschrieben wurde.

2.2.2 XML-Schema

XML-Schema als Mittel zur Definition der Grammatik von Dokumentklassen geht über die Fähigkeiten der zuvor üblichen DTDs hinaus. Es bietet neben den Beschreibungen zur Definition der Elemente und Attribute, ein Typkonzept und Strukturierungsmöglichkeiten. [Min02] Es bietet damit ein wesentlich größeres Maß an Ausdrucksmöglichkeiten zur Gestaltung von XML-Dokumenten.

Die Schemata selbst sind wiederum XML-Dokumente, die den Produktionen des Schema zur Schemadefinition genügen. Bei der Definition neuer Schemata werden deshalb alle Elemente, die dem Metaschema folgen, durch einen entsprechenden Verweis kenntlich gemacht. Dies geschieht durch Kennzeichnung des sogenannten Namensraumes. Um Schreibarbeit zu sparen, wird dieser Namensraum meist an ein lokalen Bezeichner gebunden. Dieser kann dann stellvertretend angegeben werden. Will man zum Beispiel den Namensraum des Metaschemas innerhalb eines Elements durch den Bezeichner '"xsd" ersetzen, so fügt man folgendes Attribut ein:

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

Um mit diesem Bezeichner die Zugehörigkeit eines Elements zum angegebenen Schema zu kennzeichnen, wird er einfach als durch einen Doppelpunkt vom Elementnamen getrennter Präfix verwendet. So sieht zum Beispiel die Definition eines Elements folgendermaßen aus:

```
<xsd:element name="email" type="EmailType"/>
```

Dieses Beispiel macht zusätzlich noch deutlich, dass ein Element in einem XML-Schema immer einem Typ zugeordnet wird. Dieser definiert zum einen den möglichen Inhalt des Element und zum anderen seine Attribute. Will man einen Typ nur in einem Element verwenden, so kann er auch direkt innerhalb des Elements definiert werden.

Zusätzlich zu dem `type`-Attribut gibt es noch weitere, die zum Beispiel die Häufigkeit des Auftreten eines Elements oder Standardwerte definieren. (vergl. [Min02])

Datentypen

Im abstrakten Schema-Modell sind zwei Arten von Datentypen definiert. Dies sind die einfachen- und die komplexen Datentypen. Zusätzlich existiert eine Typhierarchie, mit der sich Typen von bereits definierten ableiten lassen. Als oberstes Element der Typhierarchie oder Urtyp fungiert der Datentyp „`anyType`“. Aus ihm lassen sich durch Einschränkungen alle einfachen und komplexen Typen erzeugen. [Min02]

Einfache Typdefinitionen Alle Definitionen von einfachen Datentypen bestehen aus einer oder mehreren Einschränkungen auf Zeichenketten und Informationen über den damit beschriebenen Wert. Hierzu muss zunächst der Urtyp entsprechend ein-

geschränkt werden. Die Einschränkung für alle vordefinierten einfachen Datentypen ist „anySimpleType“. Das Metaschema leitet hiervon Typen wie `string`, `integer`, `date` oder `anyURI` ab. Die vordefinierten und die von ihnen abgeleiteten Datentypen können in allen Element- und Attribut-Deklarationen verwendet werden.

Einschränkungen können dabei auf verschiedene Arten vorgenommen werden. So kann zum Beispiel der Wertebereich durch Angabe von Intervallgrenzen eingeengt werden. Die Einschränkung aller Integer-Werte auf den Bereich zwischen 100 und 999 nach diesem Verfahren kann folgendermaßen definiert werden:

```
<xsd:simpleType name="Hunderter">
  <xsd:restriction base="xsd:integer">
    <xsd:min-inclusive value="100"/>
    <xsd:max-inclusive value="999"/>
  </xsd:restriction>
</xsd:simpleType>
```

Es können auch Muster oder Aufzählungen zur Einschränkung der einfachen Datentypen verwendet werden. Zur Erstellung eines solchen Musters werden dabei reguläre Ausdrücke verwendet, die in einer eigenen Sprache verfasst werden. Es besteht auch die Möglichkeit aus einem einfachen Datentypen eine entsprechende Liste zu definieren oder mehrere untereinander zu vereinigen. [Min02]

Komplexe Typdefinitionen Komplexe Datentypen können lediglich als Typen für Elemente verwendet werden. Sie bieten die Möglichkeit der Attribut-Deklaration, die deren Inhalt und Erscheinen regelt. Außerdem kann durch den komplexen Typ die Art und Anzahl der Kinderelemente sowie der Elementinhalt festgelegt werden.

Auch hier gibt es eine Typhierarchie, die die Ableitung eines komplexen Datentyps aus einem anderen erlaubt. Dies bedeutet, dass ein neuer komplexer Datentyp durch Einschränkung oder Erweiterung eines bestehenden Typs definiert werden kann. Gemein ist allen, dass sie vom Urtyp abgeleitet sind. [Min02]

Um ein Attribut zu einem Element hinzuzufügen, ist im komplexen Typ eine Attribut-Deklaration anzugeben. Sie definiert den Namen sowie den Bereich, aus dem der Wert stammen muss. Als Wertebereich sind lediglich einfache Datentypen zugelassen. Über zusätzliche Attribute können in der Deklaration das Erscheinen gesteuert oder Standardwerte definiert werden. Eine optionale Angabe einer URI als Attribut kann zum Beispiel durch folgendes Element in einem komplexen Typ erreicht werden:

```
<xsd:attribute name="uri" type="anyURI" use="optional"/>
```

Komplexe Typen können durch Verwendung des `simpleContent`-Elements von einfachen Datentypen abgeleitet werden. Kinderelemente werden innerhalb von strukturierenden Elementen angegeben. Ein Beispiel für ein solches Strukturelement ist die

Sequenz (`sequence`), die sagt aus, dass die enthaltenen Kindelemente in der Reihenfolge auftreten müssen, wie es im Schema definiert ist. Oder das `choice`-Element, mit dem ein gegenseitiger Ausschluss von Elementen beschrieben werden kann. (vergl. [Min02])

Beispiel

Die Schemadefinition für das in Abschnitt 2.2.1 dargestellte Element zur Beschreibung einer E-Mail-Adresse könnte wie folgt aussehen:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="email" type="EmailType"/>

  <xsd:complexType name="EmailType">
    <xsd:attribute name="type" type="xsd:string"/>
  </xsd:complexType>

</xsd:schema>
```

2.3 Entwurfsmuster

Entwurfsmuster beschreiben beständig wiederkehrende Probleme einer Domäne und fassen den Kern ihrer Lösung zusammen. Dabei werden die Entwurfsmuster dazu eingesetzt, die Entwürfe flexibler, eleganter und wiederverwendbarer zu machen. Sie stellen ein allgemeines Prinzip dar, das an die jeweilige konkrete Situation angepasst werden muss. Sie benennen ein Problem und seine prinzipielle Lösung auf einer höheren Abstraktionsebene und erweitern so das Vokabular des Softwareentwicklers. Sie helfen damit bei der Diskussion über Software und dienen auch zur einfachen Dokumentation. [GHJV96]

2.3.1 Fabriken

Die Fabrik gehört zu den objektbasierten Erzeugungsmustern. Damit stellt sie ein dynamisches Muster dar und wird eingesetzt, um in einem System Objekte zu erschaffen. Sie bietet dabei eine einheitliche Schnittstelle zum Erzeugen von Familien verwandter Objekte, ohne deren konkrete Ausgestaltung anzugeben. [GHJV96]

Klienten einer Fabrik erteilen ihr durch Aufruf entsprechender Methoden Aufträge zum Erzeugen der Objekte (siehe Abbildung 2.2). Wie diese Objekte zu erstellen sind, hängt dabei von der jeweils verwendeten Fabrik ab und ist in ihr fest vorgegeben. Die einzelnen Fabriken einer Objektfamilie haben im Allgemeinen eine gemeinsame abstrakte Fabrik als Elternklasse, die die gemeinsame Schnittstelle definiert.

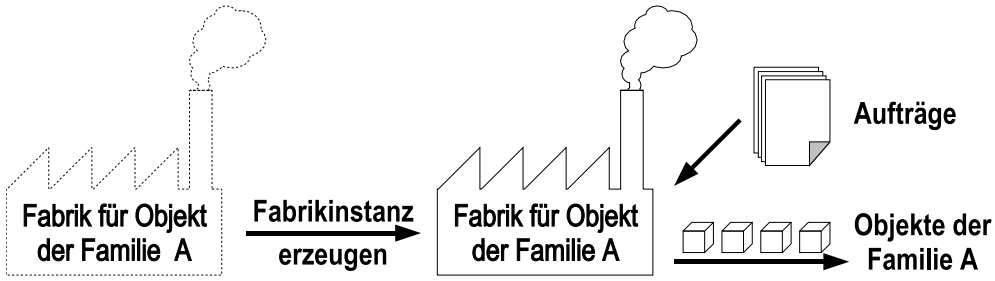

Abbildung 2.2: Arbeitsweise einer Objekt-Fabrik

Zur Verwendung kommen Fabriken, wenn man ein System erhalten möchte, das von der Erzeugung seiner Objekte unabhängig ist oder mit verschiedenen Objektfamilien umgehen soll. Aber auch dort, wo nur die Schnittstellen der zu erzeugenden Objekte bekannt sind, ist der Einsatz einer Fabrik sinnvoll. [GHJV96]

Erweiterung zur Dynamischen Fabrik

Das Entwurfsmuster der Fabrik erfährt in der onefC-Architektur eine Erweiterung zum Konzept der Dynamischen Fabrik. Diese Variante enthält zunächst keine Konstruktionsvorschrift für die zu erzeugenden Objekte. Welche konkreten Klassen zu instanziieren sind erfährt die Dynamische Fabrik erst während ihrer eigenen Instanziierung durch Übergabe entsprechender Baupläne (siehe Abbildung 2.3).

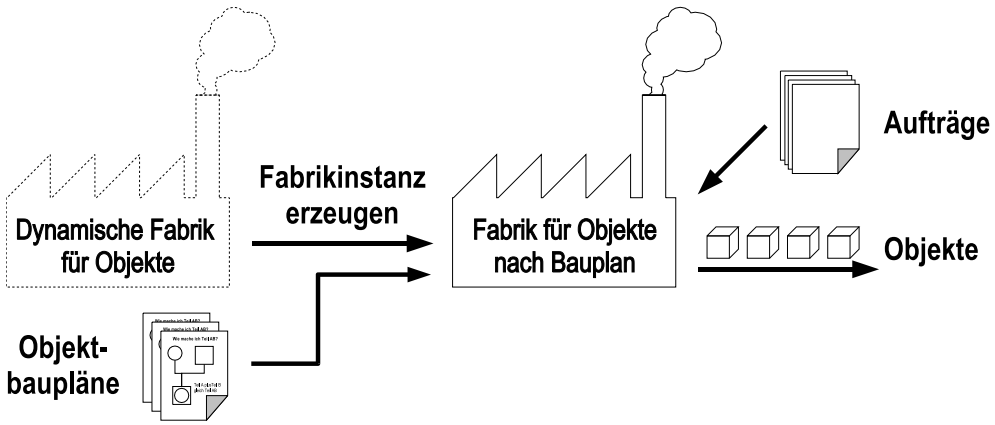

Abbildung 2.3: Arbeitsweise einer dynamischen Objekt-Fabrik

Die vom onefC-Manager verwendete `DynamicFactory` zum Beispiel liest als Bauplan kompilierte Java-Klassen ein. Dabei filtert sie die Klassen, die eine vorgegebene Schnittstelle nicht implementieren, aus. Die Motivation dieser Fabrik ist, dass sie einen Plug-In-Mechanismus realisiert. Mit ihr können durch Einstellen weiterer Baupläne und Erzeugen einer neuen Fabrikinstanz, die hinzugefügten Klassen dem System sofort zur Verfügung stehen.

2.3.2 Singleton

Auch das Singleton ist Teil der Familie der objektbasierten Erzeugungsmuster. Es wird in der Situationen eingesetzt, wo man sicherstellen möchte, dass von einer Klasse genau ein Exemplar erzeugt wird, welches von allen Klienten verwendet wird. Dabei wird der Klasse selbst die Erzeugung der Instanz und seine Verwaltung übertragen. Dazu wird die konstruierende Methode vor den Klienten verborgen und durch eine statische Klassenmethode ersetzt (siehe Abbildung 2.4). Diese Methode erzeugt bei ihrem ersten Aufruf genau eine Instanz des Objektes und speichert diese intern in einer statischen Variable. Ist das Objekt einmal erzeugt, wird durch die Exemplar-Methode lediglich eine Referenz auf die einzige Instanz übergeben. [GHJV96]

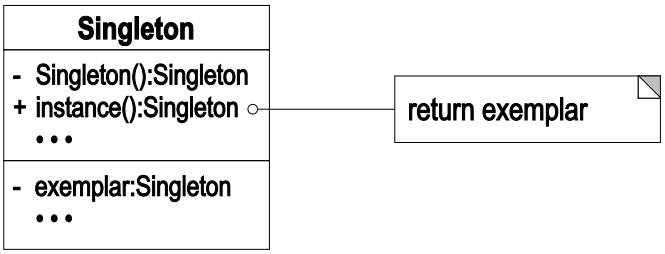

Abbildung 2.4: Struktur eines Singleton nach [GHJV96]

Singletons können immer dann angewendet werden, wenn exakt ein Exemplar einer Klasse existieren und dieses an einem speziellen Punkt ansprechbar sein soll. Auch dort, wo die einzige Instanz durch Vererbung erweitert werden soll und Klienten auf diese dann ohne Modifikation zugreifen sollen, werden Singletons eingesetzt. [GHJV96]

Nach diesem Prinzip ist zum Beispiel der `ProtocolfileMonitor` konzipiert. Durch die Gestaltung als Singleton wird sichergestellt, dass es immer nur ein Objekt gibt, dass in die Protokolldatei schreibt. Hierdurch werden Inkonsistenzen innerhalb der Daten der Datei vermieden.

2.3.3 Adapter

Der Adapter, auch bekannt als Wrapper, ist ein klassen- oder objektbasiertes Strukturmuster. Damit gehört es zu den Entwurfsmustern, die Hilfe bei der Abstimmung zwischen voneinander unabhängig konzipierter Klassen bieten. Der Adapter wird eingesetzt, um die Schnittstelle einer Klasse an die vom Klienten erwartete anzupassen (siehe Abbildung 2.5). Damit können durch die Verwendung des Adapters eigentlich inkompatible Klassen zusammenarbeiten.

Adapter werden demnach dann eingesetzt, wenn eine Schnittstelle einer zu verwendenen Klasse nicht der geforderten entspricht. Aber auch dann, wenn eine Klasse unabhängige oder nicht vorhersehbare Klassen ansprechen soll, wird dieses Entwurfsmuster eingesetzt. [GHJV96]

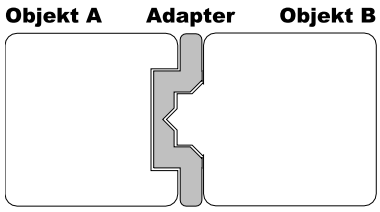

Abbildung 2.5: Anpassung einer Schnittstelle durch einen Adapter

In der onefC-Architektur wird das Adapter-Muster bei der Einbindung der Kommunikation über verschiedene Netzwerke eingesetzt. Durch Anpassen der Komponenten der unterschiedlichen Netzanbindungen an eine allgemeine Schnittstelle, kann der onefC-Manager über die verschiedenen Netzwerke kommunizieren.

2.3.4 Beobachter

Zu den objektbasierten Verhaltensmustern zählt der Beobachter. Diese Klasse der Entwurfsmuster stellt Prinzipen zur Verteilung des Verhalten zwischen Objekten auf. Damit dienen diese Muster der Gestaltung der gegenseitigen Kopplung von Objekten. Das Beobachter-Muster definiert eine 1-zu-n Beziehung zwischen Objekten. Über alle am zentralen Subjekt vorgenommenen Änderungen erhalten die abhängigen Objekte eine Nachricht. Damit sind sie in der Lage, ihren Zustand automatisch an den des zentralen Subjekts anzupassen (siehe Abbildung 2.6). [GHJV96]

Das hier gezeigte Beispiel entkoppelt ein Objekt von seiner Darstellung. Verändern sich die Werte der drei Variablen im Subjekt, so werden die unterschiedlichen Darstellungsobjekte darüber benachrichtigt und passen ihre Darstellung entsprechend an.

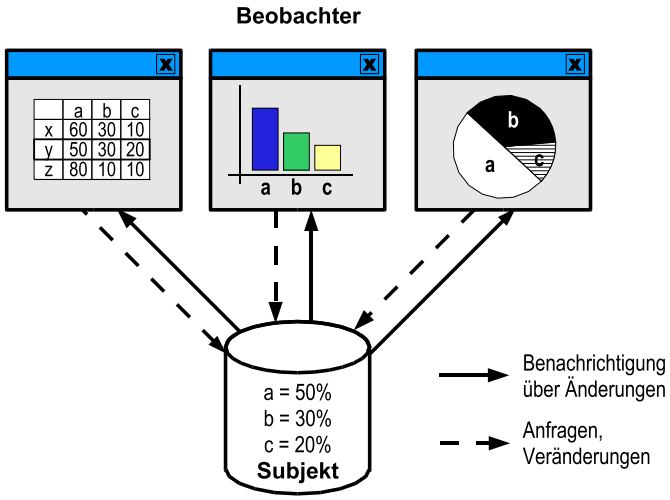

Abbildung 2.6: Prinzip des Beobachters nach [GHJV96]

Das Beobachter-Muster kann dort Verwendung finden, wo eine Konzeptionelle

Einheit zwei Aspekte umfasst, die von einander abhängen. Durch die Entkopplung in verschiedene Objekte können die beiden von einander unabhängig variiert werden und sind einzeln wiederverwendbar. Auch dort, wo die Zahl der vom Subjekt abhängigen Objekte im Vorfeld nicht feststeht, kommt das Prinzip des Beobachters zum Einsatz. [GHJV96]

Der onefC-Manager registriert sich zum Beispiel bei den Netzwerkadaptern als Beobachter, um automatisch über das Eintreffen von Identitätsnachrichten informiert zu werden.

2.4 JAVA

Für die Implementierung der onefC-Architektur als Basistechnologie für innovative Netzanwendungen, muss die verwendete Programmiersprache bestimmte Anforderungen erfüllen. Da der Identitäts-Manager möglichst einer breiten Menge von Anwendern unabhängig vom darunter liegenden Betriebssystem zur Verfügung stehen soll, muss die Umsetzung möglichst plattformunabhängig sein. Außerdem sollte die Programmiersprache die Kommunikation der Management-Komponente durch Einbindung von entsprechenden Klassen unterstützen. Darunter fallen Klassen zum Verbindungsaufbau und Nachrichtentransport aber auch Klassen zum Umgang mit abstrakten Repräsentationen von Datenstrukturen. Ein weiteres wichtiges Kriterium ist die einfache und zuverlässige Verwaltung von parallelen Programmsträngen. Eine Sprache, die diesen Anforderungen gerecht wird, ist Java von Sun Microsystems. Da diese im Projekt Verwendung findet, soll sie an dieser Stelle mit ihrem Basiskonzept vorgestellt werden.

2.4.1 Aufbau der Java Plattform

Spricht man von Java, so ist nicht nur die Programmiersprache gemeint, sondern vielmehr die Java Plattform. Die Sprache und ihre Basis-Klassen sind dabei nur ein Teil dieser Gesamtarchitektur (siehe Abbildung 2.7).

Um das Ziel der Plattformunabhängigkeit zu erlangen, haben die Entwickler der Java Plattform einen „Soft“ Computer konzipiert, der sowohl in Software als auch in Hardware realisiert werden kann. Dieses als Virtuelle Maschine (VM) bezeichnete Konzept ist so gestaltet, das es auf den existierenden Prozessoren und Betriebssystemen aufsetzen kann. Dazu wird über eine Portierungsschnittstelle und entsprechende Adapter die darunter liegenden Systeme an die Bedürfnisse der Java VM angepasst. [Kra96] Durch dieses Design, steht dem Java-Programmierer überall die gleiche Laufzeitumgebung zur Verfügung und der von ihm geschriebene Code ist universell einsetzbar. Dieses wird durch den von den Konstrukteuren geprägten Satz, „Write Once, Run Anywhere“, ausgedrückt.

Auf diesem abstrakten Computer setzt nun die eigentliche Programmiersprache mit ihren Basis-Klassen und Programmierschnittstellen auf. Die Basisklassen enthalten neben den grundlegenden Sprachelementen auch Dienste zur Dateiverwaltung,

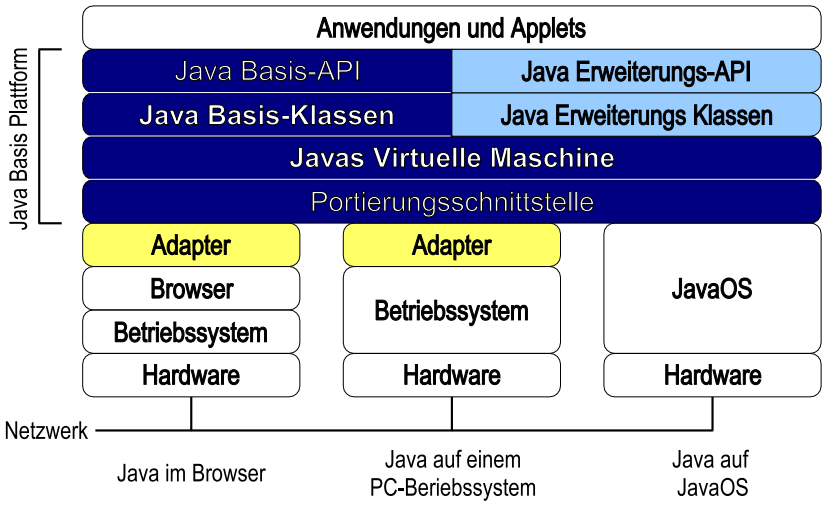

Abbildung 2.7: Java als einheitliche Plattform nach [Kra96]

zum Netzwerkanschluss oder zur Gestaltung von Benutzungsoberflächen. Damit stellt die Basis-API die minimale Voraussetzung zur Entwicklung von Java-Programmen dar. Daneben gibt es noch standardisierte Erweiterungs-Klassen, die die Fähigkeiten der Sprache ergänzen und vergrößern. Dabei bezeichnet das Wort „Standard“lediglich auf eine einheitliche Schnittstellendefinition. Die konkrete Implementierung bleibt jedoch dem Einzelnen überlassen. [Kra96]

Die Anwendungen werden mit Hilfe der APIs und der Sprachkonstrukte geschrieben und dann in sogenannten Bytecode übersetzt. Dieser stellt die Befehle für die Virtuelle Maschine dar. Der Code wird von ihr interpretiert und in entsprechende Aufrufe des darunter liegenden Betriebssystems gewandelt.

Wie in der Beschreibung der Java-Plattform zu sehen ist, erfüllt sie schon in der Basis-Version die Forderungen nach Plattformunabhängigkeit und der Anbindung an Kommunikationsnetze. Der Umgang mit abstrakten Repräsentationen von Daten, zum Beispiel als XML Datei, wird durch die Einbindung entsprechender Erweiterungen erfüllt. Und auch die Möglichkeit zur Parallelisierung von Programmteilen ist, wie im nächsten Abschnitt beschrieben wird, bereits in der Sprache enthalten.

2.4.2 Thread-Programmierung

Die Thread-Programmierung wird an dieser Stelle herausgehoben, da sie für die Erstellung einer verteilten Infrastruktur besonders wichtig ist. Die parallele Ausführung mehrerer Programmteile ermöglicht es erst, gleichzeitig mit mehreren Partnern zu kommunizieren oder einen Dienst von mehreren Klienten nebenläufig in Anspruch nehmen zu lassen.

Die Programmiersprache Java ist mit dem Ziel der Verteilung und Multithread-Fähigkeit entworfen worden. Deshalb unterstützt sie die Programmierung mit paral-

lelen, eigenständigen Prozessen direkt durch entsprechende Sprachkonstrukte. [Fla98] Dazu gehören zum einen die Klasse `Thread`, mit deren Hilfe Programmteile parallelisiert werden können, als auch Elemente zur atomaren Ausführung von Codeblöcken. Durch diese als Synchronisation bezeichnete Vorgehensweise, wird während der Abarbeitung des Blocks kein Zugriff durch andere Threads zugelassen. Dies gilt für den Code selbst als auch explizit benannte Objekte, die in dem synchronisierten Abschnitt verwendet werden. Hierdurch lassen sich Seiteneffekte der parallelen Programmausführung verhindern.

Ein neuer Thread kann dabei auf zwei unterschiedliche Arten erzeugt werden. Entweder muss die Klasse eines parallelen Programmstrangs von `Thread` erben oder sie implementiert die Schnittstelle `Runable` und wird an ein `Thread`-Objekt übergeben. In beiden Fällen wird der neue Prozess durch Aufruf der Methode `run()` gestartet. Da es nun mehrere nebenläufige Programmsequenzen gibt, muss auch die Bedingung für das Terminieren des Programms angepasst werden. In der Sprache Java ist definiert, dass ein Programm erst dann terminiert, wenn sein letzter Thread beendet wird.

Dabei kann die Parallelisierung von Aufgaben nicht nur auf Rechensystemen mit mehreren Prozessoren realisieren, sondern auch auf Einprozessor-Systemen. Dazu werden die einzelnen Threads in Teilabschnitte unterteilt, die dann abwechselnd von der Zentraleinheit bearbeitet werden. Betrachtet man jedoch die gesamte Laufzeit der Threads, so werden sie parallel ausgeführt. Da der einzelne Thread aber nicht kontinuierlich abgearbeitet wird, verlängert sich durch die hinzukommende Wartezeit seine Abarbeitung entsprechend.

In der onefC-Architektur wird die Thread-Programmierung an mehreren Stellen eingesetzt. So zum Beispiel für die im Hintergrund laufenden Server der Netzwerkkomponenten oder zur Benachrichtigung von Objekten, die sich als Beobachter (vergl. Abschnitt 2.3.4) beim Identitäts-Manager registriert haben.

2.5 Anbindung an TCP/IP

Betrachtet man eine Identitäts-Infrastruktur bezüglich ihrer Anbindung an ein Kommunikationsnetz, so erkennt man, dass sie als Teil der Anwendungsschicht auf die Transportschicht zugreift (siehe Abbildung 2.8). Diese Anbindung soll folgend am Beispiel des Internet-Protokollstapels gezeigt werden.

2.5.1 Aufbau der Protokollhierarchie im Internet

Die Hierarchie der Protokolle im Internet kann wie das OSI-Referenzmodell in Schichten eingeteilt werden. Jedoch besitzt der nach den zwei dominierenden Protokollen auch als TCP/IP benannte Protokollstapel lediglich vier Schichten (siehe Abbildung 2.8). Dabei ergibt sich die Reduktion durch eine Zusammenfassung der Bitübertragungs- und Sicherungsschicht sowie dem Fehlen der Sitzungs- und Darstellungsschicht. [Tan97] Übrig bleiben somit die Anwendungsschicht, die Transportschicht,

die Vermittlungsschicht und die kombinierte Übertragungs- und Sicherungsschicht (auch Host-an-Netz-Schicht genannt).

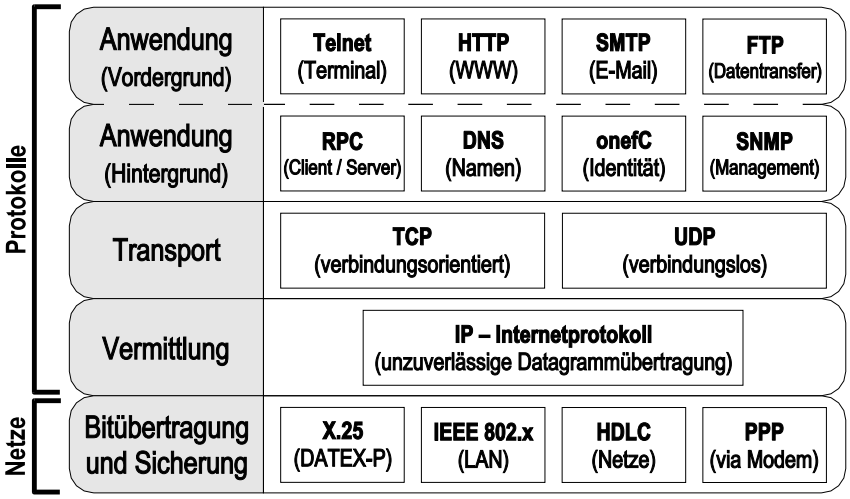

Abbildung 2.8: Der Internet-Protokollstapel nach [Lip98, Tan97]

Die Host-an-Netz-Schicht

Die Spezifikation der TCP/IP-Protokollhierarchie sagt über diese Schicht recht wenig aus. Es wird lediglich vorausgesetzt, dass der Host an ein Kommunikationsnetz angeschlossen ist, über das Pakete des Internetprotokolls (IP) übertragen werden können. Es ist also kein allgemeines Protokoll definiert. Somit kann es von Host zu Host abweichen, je nachdem, welcher Netztyp verwendet wird. [Tan97] Diese können zum Beispiel lokale Netze (LAN) nach IEEE 802.x oder globale Netze über X.25 oder ATM sein. Aber auch Punkt-zu-Punkt-Verbindungen über Modem oder ISDN sind möglich. [Lip98]

Die Vermittlungsschicht

Auf der Vermittlungsschicht besteht keine Wahl, hier ist das Internet Protokoll zu verwenden. Es handelt sich hierbei um ein unzuverlässiges und paketvermittelndes Protokoll. IP schickt dabei seine Datagramme verbindungslos über die darunter liegende Vernetzungsschicht, sogar über Grenzen verschiedener Netze hinweg. [Tan97] Dabei werden die einzelnen Paket als unabhängig betrachtet und einzeln transportiert. Dies kann dazu führen, dass manche Pakete einen anderen Weg zum Empfänger nehmen oder einzelne Datagramme verloren gehen. Die Größe der IP-Pakete kann dabei zwischen 576 Bytes und 65.535 Bytes liegen. Innerhalb dieser Bandbreite können die Datagramme auf ihrem Weg beliebig geteilt oder vereinigt werden. [Lip98]

Die Adressierung von Quell- und Zielrechner geschieht über 32 Bit lange IP-Adressen. Jede dieser Adressen identifiziert einen Rechner im globalen Netzverbund.

[Lip98] Auf Anwendungsebene gibt es durch das Domain Name System (DNS) die Möglichkeit Namen IP-Adressen zuzuordnen. Durch diesen Dienst wird die Adressierung auf der Anwendungsebene vereinfacht. So sieht zum Beispiel die Zuordnung von Name und IP-Adresse des Web-Servers der Universität Hamburg folgendermaßen aus :

$$\text{www.uni-hamburg.de} \Rightarrow 134.100.32.72$$

Dabei ist zu beachten, dass es sich nicht um eine eins-zu-eins Beziehung handelt. Es ist sehr wohl möglich, dass mehrere Namen zu einer Adresse existieren.

Zur Vermittlungsschicht werden auch noch die Protokolle ICMP für Fehlermitteilungen und Steuerinformationen sowie IGMP für Gruppenkommunikation gerechnet. Jedoch basieren diese auf dem IP-Protokoll und sind deshalb nicht als eigenständiges Vermittlungsprotokolle anzusehen. [Lip98]

Die Transportschicht

Auf der Transportschicht sind zwei Ende-zu-Ende Protokolle definiert. Dies ist zum einen das Transmission Control Protocol (TCP) als ein zuverlässiges und verbindungsorientiertes Protokoll und zum anderen das User Datagramm Protokoll (UDP) als unzuverlässiges und verbindungsloses Protokoll.

TCP ist in der Lage einen Bytestrom fehler- und verlustfrei von einem Rechner auf einen anderen zu übertragen. Dazu teilt es die zu übertragenden Daten in einzelne Nachrichten auf und leitet diese über die Internet-Schicht weiter. Die TCP-Schicht des Empfängers verknüpft die empfangenen Nachrichten wieder zur Originalnachricht. Sollten Datenpakete auf der Vermittlungsschicht verloren oder verfälscht worden sein, so werden die entsprechenden Nachrichten erneut vom Sender angefordert. Auf dieser Ebene ist auch ein Mechanismus zur Flusssteuerung angesiedelt, um die Überlastung eines langsamen Empfängers zu verhindern. [Tan97]

UDP hingegen ist datagrammorientiert und enthält keine Mechanismen, um die Zustellung zu garantieren. Es wird dort eingesetzt, wo die Abfolge- und Flusskontrolle von TCP nicht notwendig ist oder selbst vorgenommen werden soll. Auch dort, wo es auf Geschwindigkeit und nicht so sehr auf Genauigkeit ankommt, wird UDP eingesetzt. Dies ist zum Beispiel beim Audio- oder Videostreaming der Fall. [Tan97]

Die Anwendungsschicht

In der Anwendungsschicht werden die höheren Protokolle zusammengefasst. [Tan97] Sie können in Vorder- und Hintergrunddienste eingeteilt werden. Dabei werden diejenigen, die eine Benutzungsoberfläche haben als Dienste im Vordergrund bezeichnet. Dies sind zum Beispiel der Austausch elektronische Post via SMTP, der Dateitransfer mit FTP oder das Browsen im World Wide Web. Zu den Hintergrunddiensten, also denjenigen, auf denen andere Anwendungen aufsetzen, gehören zum Beispiel der Remote Procedure Call, das Domain Name System oder Netz Dateisysteme (NFS).

[Lip98] In dieser Schicht kann auch die onefC-Identitäts-Infrastruktur angesiedelt werden, da sie über TCP an das Internet angeschlossen ist.

2.5.2 Etablieren einer Verbindung über die Transportschicht

Zur Etablierung einer Verbindung zwischen zwei Anwendungen über eine TCP Verbindung, genügt die Angabe der Rechneradresse nicht. Vielmehr muss noch zusätzlich eine noch der konkrete Transportdienst-Übergabepunkt der Daten, der sogenannte Port, bekannt sein. Zusammen bilden Adresse und Port einen Socket, der als Verbindungsendpunkt dient. Demnach besteht die konkrete Verbindung aus dem Paar der Sockets der beiden kommunizierenden Einheiten. Eine solche Verbindung ist immer Vollduplex und Punkt-zu-Punkt ausgelegt. Dass bedeutet, dass Datenpakete in beide Richtungen gleichzeitig gesendet werden können. Jedoch ist es nicht möglich mit einem Paket gleichzeitig mehrere Empfänger zu adressieren. [Tan97]

Um eine Verbindung zwischen zwei Prozessen einzurichten muss sich einer als Server an der Transportschicht anmelden. Als Server signalisiert der Prozess sein Interesse an den Daten, die an dem von ihm angegebenen Port ankommen. Der zweite Teilnehmer stellt als Client eine Socketverbindung zum Socket des Servers her. Dabei wird der Port des Clients durch das Betriebssystem bestimmt. Die nun etablierte Verbindung kann nun zum Datenaustausch genutzt werden. Um die Verbindung zu beenden, muss einer der Sockets geschlossen werden.

Um gezielt Dienste wie FTP oder TELNET ansprechen zu können, sind für eine Vielzahl von Anwendungen die Portnummern standardisiert worden. So kann man im Allgemeinen FTP über den Port 21 und TELNET über 23 erreichen.

Kapitel 3

Analyse des Identitäts-Begriffs

In diesem Kapitel sollen die grundlegenden Begriffe dieser Arbeit betrachtet werden. Dazu wird der Begriff der Identität und des Identitäts-Management untersucht. Hieraus werden grundlegende Anforderungen an eine entsprechende Architektur formuliert. Abschließend werden bestehende Ansätze des Identitäts-Managements betrachtet und vergleichend analysiert.

3.1 Identitäts-Begriff

In diesem Abschnitt soll der Begriff der Identität unter den Gesichtspunkten seines zwischenmenschlichen Verständnisses betrachtet werden. Dies erfordert die Analyse der Identität unter verschiedenen Aspekten, die ihren Ursprung in der Philosophie, Psychologie und Soziologie haben. Konkret geht es dabei um die Eigenschaft der Identität als Instrument zur Identifizierung, als Ausdruck der Persönlichkeit und als Teil von Gruppenidentitäten. Zusätzlich wird auf die eher technischen Aspekte des Umgangs mit der Identität in der Informatik und in Staat und Gesellschaft eingegangen.

3.1.1 Philosophische Einordnung

Betrachtet man die Identität als zweistellige Relation zwischen Gegenständen beliebiger Bereiche, so ist sie dadurch ausgezeichnet, dass jeder Gegenstand allein zu sich selbst in Beziehung steht. Die Identität ist damit die feinste denkbare Äquivalenzrelation[1] und somit Grenz- oder Spezialfall der Gleichheit. Deshalb wird sie auch als totale, vollständige oder absolute Gleichheit bezeichnet. Diese vollkommende Übereinstimmung, das heißt Gleichheit in allen Eigenschaften oder Hinsichten, führt dazu, dass Identisches immer auch Gleiches ist. Die Umkehrung dieser Aussage ist jedoch nicht erlaubt. [Mit84, BRO89b] Die sprachliche Unterscheidung von *„dasselbe“* und

[1] Äquivalenzrelationen sind Relationen, die symmetrisch, reflexiv und transitiv sind.

„das Gleiche“, die diesen Sachverhalt ausdrückt, ist jedoch aufgegeben worden (vergl. [Dud96, Seite 318]).

Neben der abstrakte Definition als Relation zwischen Gegenständen gibt es seit Aristoteles[2] den umstrittenen Versuch die Identität über die Ununterscheidbarkeit zu definieren. Dabei werden Dinge als identisch angesehen, wenn alle Aussagen des einen auch von dem anderen ausgehen. [Mit84] Gottfried Wilhelm Freiherr von Leibniz[3] fasst diesen Ansatz in seiner Definition der numerischen Identität auf. In ihr definiert er die Identität so restriktiv, dass jegliche Verschiedenheit – auch in den Zuständen des Objekts – ausgeschlossen wird. Dies bedeutet, dass zwei Objekte in keiner Eigenschaften abweichen dürfen, um identisch zu sein. Denn *„solche Objekte können deshalb [...] numerisch identisch genannt werden, weil sie in Wirklichkeit nur ein einziges Objekt sind, das unter verschiedenen Gesichtspunkten betrachtet wird.“* [Hen76, Seite 76f]

Leibniz strikte Begriffsbildung wurde früh unter anderem von Christian August Crusius[4] kritisiert. Ein Kritikpunkt war, dass Leibniz den Identitätsbegriff willentlich überdehnt. Da der Begriff der Identität jedoch eine Beziehung zwischen verschiedenen Zuständen eines Objektes herstellt, darf er nicht dahingehend umdefiniert werden, dass der Wechsel in andere Zustände durch eben diesen Begriff verboten wird.

Die Kritik an der strikten numerischen Identität hat zur Herausbildung des Begriffs der gemäßigten numerischen Identität geführt. Hierbei bleibt die Identität eines Objektes auch dann erhalten, wenn trotz eines Wechsels der Zustände konstitutive Eigenschaften erhalten bleiben. Die Identität besteht aber auch, solange in einem kontinuierlichen aber niemals totalen Wechsel das Objekt ununterbrochen fortbesteht. [Hen76]

3.1.2 Psychologische Einordnung

Identität des Individuums

Der Begriff der Identität wird in der Psychologie häufig als Kurzbezeichnung für die Selbst- oder Ich-Identität verwendet. [BRO89b] Die Frage nach dieser Form der Identität wird seit Sigmund Freud[5] gestellt. Dabei ist die Identität eines Menschen mit sich selbst gemeint, also die Konstruktion des einzelnen Individuums oder eines Individualbereiches. [Mit84] Klassische Konzepte erklären dies mit dem *„Bewusstsein, sich von anderen Menschen zu unterscheiden (Individualität), sowie über die Zeit (Kontinuität) und verschiedene Situationen (Konsistenz) hinweg [...] dieselbe Person zu bleiben.“* [Dör99, Seite 255] Die Identitätsbildung basiert dabei auf wechselseitigen Erfahrungen und Beziehungen bei der Übernahme von Rollen in unterschiedlichen sozialen Kontexten. Diese Identitätsentwicklung führt schrittweise zur Integration in ein stabiles und einheitliches Ganzes. Dieses Konstrukt, das sich zur Identität verfe-

[2] griechischer Philosoph, * 384 v. Chr. Stageira (Thrakien), † 322 v. Chr. Chalkis

[3] deutscher Philosoph, * 1.7.1646 Leipzig, † 14.11.1716 Hannover

[4] Philosoph und Theologe, * 10.6.1715 in Leuna bei Merseburg, † 18.10.1775 in Leipzig

[5] österreichischer Psychiater, * 6.5.1856 Freiberg, Mähren, † 23.9.1939 London

stigt, wird auch als *Subjektives Selbst* oder *I* bezeichnet und beinhaltet Aspekte des Selbstempfinden und der Selbstevidenz. Die Erfahrung des eigenen Willens und der persönlichen Kontrolle führen zur Empfindung der Eigenbestimmung und der Abgrenzung zu anderen Personen und Dingen. Die angesprochene Erfahrung der Konsistenz und Kontinuität begründen das Gefühl der Einheitlichkeit im Zeitverlauf. Es wird also im traditionellen Identitätsverständnis die Dauerhaftigkeit und Einheit des Selbst betont. [Tur99, Res98, Dör99]

Diese Konzentration auf ein einheitliches Ganzes wird jedoch dem Umstand nicht gerecht, dass sich die Identität gerade durch die Übernahme einer Vielzahl unterschiedlicher Rollen ausbildet. Dabei wird in den verschiedenen Umgebungen die eigene Charakteristik in Paketen verschiedener Größe und Inhalt präsentiert. Deshalb fassen neuere Konzepte die Identität als eine komplexe Struktur auf, die aus einer Vielzahl von Elementen besteht (Multiplizität). Teilmengen dieser Elemente werden dann kontextabhängig aktiviert oder deaktiviert (Flexibilität). Das bedeutet, dass es nicht eine „einzige“ oder „wahre“ Identität gibt, sondern viele Gruppen-, Rollen- , Körper- oder tätigkeitsbezogene Teil-Identitäten. Die Entwicklung der Identität wird als ein lebenslanger Prozess angesehen, der durch tägliche Identitätsarbeit immer neue Teil-Identitäten in ein „Patchwork“ oder „Pastiche“ einfügt. [Sul96, Dör99]

Die Inhalte der Teil-Identitäten basieren dabei auf unterschiedlichen Aspekten des Selbst. Nicht nur körperliche Attribute, wie Größe, Alter, Geschlecht oder Aussehen, sondern auch Fähigkeiten im Vergleich mit anderen, die soziale Ausstrahlungskraft und die eigenen Gefühle sind enthalten. Dabei wird die Fähigkeit neue Teil-Selbste in Abhängigkeit des aktuellen Kontextes zu bilden, als Identitäts-Kompetenz bezeichnet. Diese Identitätsfragmente werden miteinander in Beziehung gesetzt, was auch eine partielle Abschirmung einschließen kann. [Res98, Dör99]

Die Dauerhaftigkeit und Einheit, die der traditionellen Identitätsbegriff betont, kann auch in die komplexen modernen Konzepten integriert werden. Denn Teil-Identitäten können so verstanden werden, dass es gemeinsame Identitäts-Kerne bzw. übergeordnete Meta-Identitäten gibt. [Dör99]

Selbstdarstellung Neben der privaten Dimension der Identität gibt es auch eine öffentliche – der nach außen präsentierten Aspekte des Selbst. Dabei ist die Darstellung der Identität immer ein Balanceakt zwischen sozialen Normen und den Ansprüchen, die die Person an sich selbst stellt. Da Personen sich zu sich selbst verhalten können, werden die präsentierten Inhalte entsprechend der gerade eingenommenen Rolle gewählt.Gerade bei Attributen, die für den Kommunikationspartner nicht nachprüfbar sind, besteht die Möglichkeit, diese zu verändern oder zu erfinden. Dabei gestalten wir unser Verhalten so, dass ein zielkonformer Eindruck entsteht. Dieser Eindruck muss dabei nicht immer positiv sein. [Fuc02, JP00, Dör99]

Dieser (oft unbewusste) Wechsel der offenbarten Teil-Identität, also der rollen- oder situationsabhängigen Ausprägungen der präsentierten Identität, wird als Prozess der Selbstdarstellung bezeichnet. Dabei kann eine beliebig große Kluft zwischen

bürgerlicher Rolle und privatem Selbst eintreten. [JGM01, Fuc02] Dies führt dazu, dass bei jedem Kommunikationspartner ein eigenes Bild bzw. Image von der Person entsteht (siehe Abbildung 3.1).

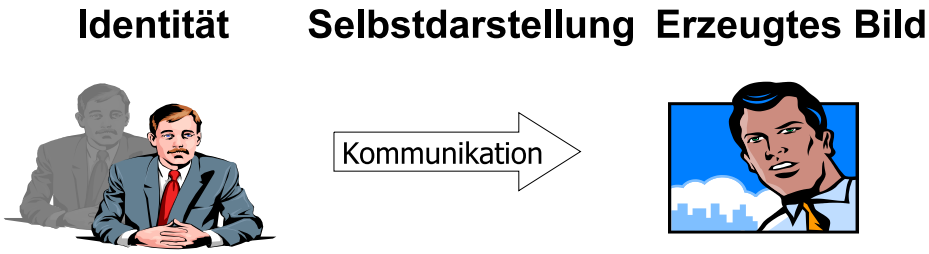

Abbildung 3.1: Selbstdarstellungs-Prozess

Die Selbstdarstellung wird von verschiedenen Faktoren beeinflusst. So weicht zum Beispiel öffentliches Verhalten von privatem ab. Auch der Adressat, die Intensität des Kontakts und die Intention beeinflussen den Darstellungsprozess. Da das Phänomen der Selbstdarstellung ein ubiquitäres in der Gesellschaft ist, wird ihm bei der Einschätzung eines Anderen meist schon unbewusst Rechnung getragen. [Dör99]

Netzidentität Die Netzwelt ist für Viele vor allem auch ein sozialer Raum, in dem der Austausch von Ideen, Gedanken und Emotionen einen Kontext aufbaut. Die dabei entstehenden virtuellen Identitäten stehen im Zusammenhang mit der Art, wie sich das Individuum im Netz präsentiert und wie es mit Anderen kommuniziert. Die Identitäten können dann ausschließlich auf Erfahrungen, Beziehungen und Interaktionen in Computer-Welten basieren. Dabei ist es möglich reale Attribute durch fiktive zu ersetzt. Dies kann soweit gehen, dass eine virtuelle Teil-Identität mit der realen nichts oder nur sehr wenig gemein hat. Zwischen der Modifikation einzelner Attribute und der Konstruktion einer neuen virtuellen Person besteht dabei ein fließender Übergang. Die erschaffene Online-Identität kann durchaus ein Eigenleben haben. Andererseits fließen oft unbewusst oder aber ganz bewusst reale Selbst-Aspekte in die Gestaltung ein. Ein Individuum kann so viele elektronische Persönlichkeiten kreieren wie es möchte. Oftmals ist der gemeinsame Erzeuger zwischen ihnen die einzige Verbindung, die diese besitzen. Es gibt sogar Gruppen im Cyberspace, die das kreieren einer neuen Persönlichkeit fordern und fördern. So ist in MUDs[6] und anderen Online-Spielen das Kreieren einer neuen Fantasie-Identität eine Selbstverständlichkeit. [Deb96, JP00, Dör00, Tur99, Dör99, Don96, Sul96]

Das Erschaffen von willkürlichen Identitäten wird durch die im sozialen Raum des Internets bestehende medienspezifische Kanalbegrenzung begünstigt. Im Gegensatz zur direkten Kommunikation, fehlt es dort an Verbindlichkeit und Nachprüfbarkeit.

[6] Abkürzung für *Multi User Dungeon*: onlinebasiertes Rollenspiel

So werden Attribute, die der Charakteristik des gewählten Kommunikationskanals entsprechen, verstärkt, andere Fähigkeiten oder Unzulänglichkeiten bleiben hingegen außen vor. Obwohl die Online-Identität jeweils eine Rolle darstellt, spielt das Individuum doch sich selbst, da das neu entworfene Bild Teil des Identitäts-Patchwork wird. [Deb96, JP00]

Verteilung von Identität

Betrachtet man Identitätskonzepte unter dem Gesichtspunkt der Verteilung, so kann man zwei Aspekte unterscheiden. Zum einen die Ausbildung einer gemeinsamen Identität in Gruppen und zum anderen die Verbreitung von Teil-Identitäten in Kommunikationsbeziehungen.

Soziale Identitäten Neben der Ausbildung der persönlichen, individuellen Identität entstehen in Gruppen auch sogenannte soziale Identitäten. *„Soziale Identität bezieht sich [...] auf eine soziale Kategorie einer ungeordneten Menge von Personen, die [...] sich in der Regel bezüglich der Entscheidung über eine Mitgliedschaft und (angeblich) charakteristischen Eigenschaften oder Attributen unterscheiden.“ [AH01, Seite 7]* Dieses Gefühl der Zugehörigkeit zu einer Gruppe wird auch als „Ingroup“ bezeichnet.

Die Eigenschaften, die eine soziale Identität definieren, unterscheiden sich nicht von denen, die eine personale Identität bilden. Dies führt dazu, dass der Übergang von individueller zur Gruppen-Identität fließend ist. Dieser Umstand begründet auch, dass eine Entfaltung in einer Gruppe auch immer eine Entwicklung der personalen Identität ist. Eine wichtige Eigenart muss beim Umgang mit sozialen Identitäten jedoch beachtet werden. Nicht jeder, der sich „Ingroup“ fühlt, muss mit allen Attributen der Gruppe sympathisieren. Es muss aber ein gemeinsames Fundament bestehen, welches die Gruppe zusammenhält. [Dör99, Deb96, Don96]

Durch Anonymität, wie sie zum Beispiel im Internet herrscht, verstärken sich die sozialen Identitäten, da individuelle Eigenschaften in den Hintergrund treten. Dies führt dazu, dass die Gruppe homogener scheint und konkurrierenden individuellen Merkmalen weniger Gewicht beigemessen wird. Die Anonymität fördert daneben auch die Entwicklung nur latent vorhandener Teile des Identitäts-Patchworks. [Dör99]

Verteilte Abbilder der Identität Gibt man während einer Kommunikation Teile seiner Identität preis, so entsteht beim Gegenüber ein Bild der eigenen Person (vergl. Abschnitt 3.1.2). Dieses einmal offenbarte Wissen kann dann nicht mehr zurückgenommen werden, solange die Teil-Identität mit dem Individuum verknüpft werden kann. Diesen Umstand bezeichnet man auch als Monotonie der Anonymität. [JG01b] Über verschiedene Kommunikationsbeziehungen hinweg entstehen so mehrere sich in der Attributmenge unterscheidende Bilder. Wenn es möglich ist die einzelnen Teil-Identitäten durch identifizierende Attribute miteinander zu verketten, kann ein

detaillierteres Bild der Person erzeugt werden. Dieses Zusammenfügen der einzelnen Teil-Identitäten kann ohne das Wissen des Identitätsinhaber geschehen.

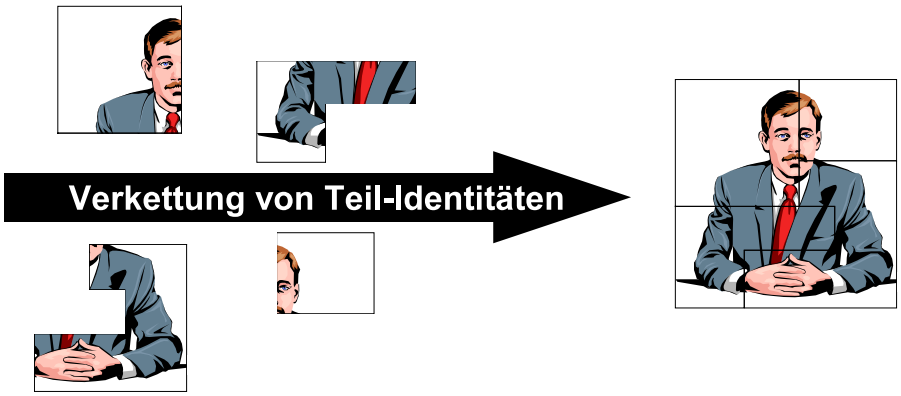

Abbildung 3.2: Verknüpfung von Teil-Identitäten

Ist man sich dieser Situation beim Umgang mit seiner Identität bewusst, so kann man diese Tatsache bei der Kommunikation berücksichtigen. Auf der einen Seite muss man, wenn man die Verknüpfung erschweren oder verhindern möchte, mit identifizierenden Attributen vorsichtig umgehen. Auf der anderen Seite kann man sich diesen Umstand auch zu Nutze machen, indem man dem Kommunikationspartner explizit erlaubt gewisse Daten weiterzugeben. In wie weit sich dieser jedoch an Weitergabever- und -gebote hält, steht außerhalb des Einflusses des Identitätsbesitzers.

3.1.3 Technische Einordnung

Identität in der Informatik

Der Begriff der Identität ist in der Informatik nicht unbekannt. Er wird jedoch meist in seiner Bedeutung als Äquivalenzrelation verwendet. Also zur eindeutigen Wiedererkennung eines Individuums oder Objektes in verschiedenen Situationen oder Kontexten.

So spricht man zum Beispiel in Public Key Infrastrukturen (PKI) von digitalen Identitäten. Dabei ist jedoch nicht ein Äquivalent zum alltäglich gebrauchen Begriff gemeint sondern wesentlich weniger. Hier bezeichnet der Begriff der Identität lediglich einen elektronischen Ausweis in Form eines Zertifikates. [DT02] Eine solche elektronische Bescheinigung hat die Aufgabe einen öffentlichen- oder Prüfschlüssel einer Person zuzuordnen und deren Identität zu bestätigen. [Sig01, §2(6)] In Verbindung mit dem geheimen Schlüssel der Person, kann diese dann Transaktionen ausführen, die ihr eindeutig zugeordnet werden können.

Auch in Programmiersprachen gibt es die Unterscheidung zwischen Gleichheit und Identität. In der Programmiersprache JAVA kann man zum Beispiel zwei Variablen mit

Hilfe des Boolschen-Operators „==“ darauf prüfen, ob sie ein und dasselbe Objekt referenzieren. Will man hingegen wissen, ob die zwei referenzierten Objekte gleichen Inhalts sind, kann die jedem Objekt eigene Methode `equals()` verwendet werden. Hierbei zeigt sich wieder, dass die Identität ein Spezialfall der Gleichheit ist, denn überall dort wo die Identitätsprüfung erfolgreich ist, ist auch die Methode zur Überprüfung der Gleichheit wahr. Jedoch gilt die Umkehrung dieser Aussage nicht (vergl. Abbildung 3.3). Die Feststellung der Identität, kann auf einem Rechner über die eindeutige Adresse der Objekte im Speicher geführt werden. In verteilten Systemen kann dieses Kriterium nicht verwendet werden, da die Objekte nicht im lokalen und somit eindeutigen Adressraum des Rechners vorliegen müssen. In solchen Fällen können global eindeutige Bezeichner den Objekten zugeordnet werden, um eine eindeutige Eigenschaft für die Identifizierung zu erhalten.

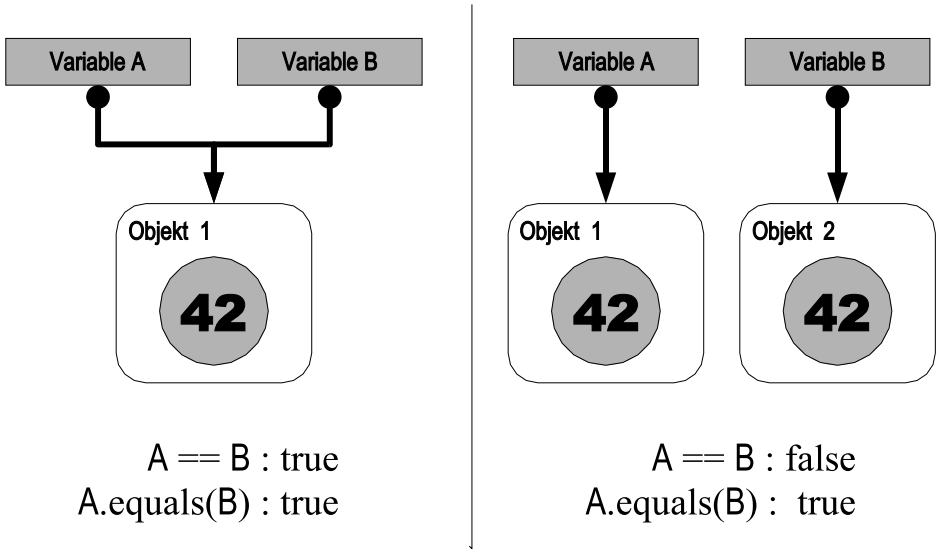

Abbildung 3.3: Unterschied zwischen Identität und Gleichheit in JAVA

Identität in Staat und Gesellschaft

In dieser Domäne wird die Identität meist als Zugriffsmethode auf eine Person gesehen. Sie wird benutzt, um die mit der Person assoziierten Parameter, Eigenschaften oder Attribute zu bestimmen, um sicherzustellen, dass es sich um das korrekte Individuum handelt. Die meisten der heute von behördlicher Seite verwendeten Schemata können unter dem Begriff der Ausweis-Identität subsummiert werden. Diese stellen eine Sammlung von Eigenschaften der Person da, wie zum Beispiel Name, Nationalität, Geburtsort und -tag. [Cap01]

Ist eine zweifelsfreie Identifizierung zu führen, kann das Kriterium der kontinuierlichen Existenz im Raum angewendet werden (vergl. Abschnitt 3.1.1). Da körperliche Dinge sich nicht zur selben Zeit an verschiedenen Orten aufhalten können, reicht es

aus ihre Spur bis zum gewünschten Zeitpunkt zurück zu verfolgen, um die Identität zu folgern. So wird beispielsweise die Kriminaltechnik dazu verwendet, herauszufinden, ob ein Projektil dasselbe ist, welches sich zuvor im Lauf einer bestimmten Waffe befunden hat. [Wai00]

Lässt sich dieser Beweis jedoch nicht führen, kann man versuchen konstitutive Eigenschaften zu finden, die sich nicht verändern. Dabei ist jedoch deren Aussagekraft zu beachten. Dies führt im Allgemeinen dazu, dass die Identität nur mit einer gewissen Wahrscheinlichkeit angegeben werden kann. Zu diesen Eigenschaften zählt zum Beispiel der genetische Fingerabdruck eines Lebewesens. Die Wahrscheinlichkeit der Übereinstimmung zwischen zwei Menschen beträgt etwa $1 : 10^{10}$. Ausnahmen sind eineiige Zwillinge. Diese besitzen sogar den selben genetischen Fingerabdruck. Durch eine Linearkombination von Eigenschaften kann man deren Aussagekraft wesentlich erhöhen. Dabei wird die Schnittmenge aller Personen immer weiter eingeschränkt, bis die Identitäts-Wahrscheinlichkeit ein Sicherheitsniveau erreicht. Dabei drückt die Wahrscheinlichkeit das Maß der graduellen Abstufung zwischen Anonymität und Information über die Identität aus (vergl. [Köh00]).

3.2 Identitäts-Management

Das soziale Verhalten der kontext-, rollen- und situationsabhängigen Präsentation der eigenen Identität ist die Inspiration des Identitäts-Managements. [BK00] Somit kann es als digitales Äquivalent der Selbstdarstellung bezeichnet werden. Dies verdeutlicht, dass beim Identitäts-Management der Benutzer und seine Bedürfnisse im Mittelpunkt stehen.

Ein Identitäts-Manager stellt dabei ein universelles Sicherheitswerkzeug zur Steigerung des Schutzes persönlicher Daten dar (vergl. Tabelle 3.2). Die Entscheidung, wer in einer Situation welche Daten erhält, soll durch den Benutzer selbst bestimmt werden können. So sollte dieser zum Beispiel die Möglichkeit haben, zu entscheiden, dass ein Online-Shop seine Kreditkartennummer erst zu dem Zeitpunkt bekommt, wenn die Transaktion dies erfordert. Dies könnte so aussehen, dass während der Informationsphase der Zugriff auf diese Daten verweigert wird und erst bei der Abwicklung des Kaufes würden sie dann freigegeben werden. Durch diese Möglichkeit der Steuerung der Datenverbreitung wird der Nutzer bei der Trennung verschiedener Lebensbereiche und dem Agieren in unterschiedlichen Rollen unterstützt. Außerdem wird der Benutzer durch die Befähigung zum Selbstschutz gegenüber dem Kommunikationspartner gestärkt. Dabei müssen die Entscheidungen nicht prinzipiell manuell getätigt werden, sondern können auch auf Regeln aufbauen, die der Nutzer zuvor definiert hat. Dies kann soweit gehen, dass für jede Situation die Zugriffsrechte automatisch neu ausgehandelt werden. [JGM01, BK00, Bor96, JKZ02, Köh00, JG00]

Eine weitere Aufgabe des Identitäts-Managements ist es, durch eine Zusammenführung möglichst aller Informationen eines Profils, eine Fragmentierung der Identität zu verhindern. Dies erleichtert die Wartung der Daten und führt somit zu einer

Steigerung der Qualität und Aktualität. Dies wiederum erhöht den Nutzen der Daten für diejenigen, die diese verarbeiten. So würde zum Beispiel die Änderung der Telefonnummer sofort in allen folgenden Transaktionen berücksichtigt werden und der Dienstanbieter hätte einen Gewinn an Aktualität der von ihm verarbeiteten Daten. Die Konzentration der Identitätsdaten beim Nutzer hat auch den Vorteil, dass dieser durch Protokollierung der Kommunikation nachvollziehen kann wer welche Daten bekommen hat. [CMBG+02, CMB02, BK00, JG01a, Köh00]

Es ist außerdem die Aufgabe des Identität-Managements die Authentifizierung des Benutzers in verschiedenen Kontexten zu ermöglichen. Dazu muss eine eindeutige und über die Zeit konsistente Identifizierung möglich sein. Dies ist notwendig, da der Prozess der Authentifizierung feststellen soll, ob der Identitäts-Inhaber dem Partner bereits bekannt ist. [Cla99] Die Authentifizierung ist zum Beispiel dann notwenig, wenn den einzelnen Benutzern unterschiedliche Rechte oder Dienste zugeordnet werden sollen.

Jedoch darf auch die Identifizierbarkeit nur dann gegeben sein, wenn dies vom Nutzer des Managers erlaubt ist. Diese Forderung impliziert, dass in den Fällen, wo eine Identifizierbarkeit nicht gewünscht ist, eine Pseudonymisierung stattfinden muss. Die Verwendung einer pseudonymisierten Identität verhindert jedoch nur die direkte Identifizierung. Häufig ist es nämlich möglich, durch die in der Identität enthaltenen Attribute indirekte Rückschlüsse auf den Benutzer vorzunehmen. Eine solche indirekte Identifizierung ist jedoch nicht immer eindeutig, sondern mit einem Risiko behaftet. Deshalb spricht man in solchen Fällen dem Identitäts-Inhaber einen gewissen Grad an Anonymität zu. Die Identifizierung ist dann nur mit einem bestimmten Sicherheits-Niveau gegeben. [BK00] Es gibt also zwischen vollständiger Anonymität und eindeutiger Identifizierung eine große Bandbreite von Abstufungen.

Der Identity-Manager hat auch die Aufgabe den sogenannten Identitäts-Diebstahl zu verhindern. [Köh00] Dabei wird der Begriff des Diebstahls nicht in Bezug auf die Entwendung von Attributen aus dem Manager verstanden. Vielmehr ist die Situation gemeint, dass Fremde sich mit denen ihnen anvertrauten Daten als der Identitäts-Inhaber ausgeben. Es ist also das Vortäuschen einer fremden Identität gemeint.

Ein Identitäts-Management System soll den Nutzer also dabei unterstützen, Identitäten zu definieren und die in ihnen enthaltenen Daten entsprechend der eingenommenen Rolle zu präsentieren. Dabei soll ein möglichst breites Spektrum zwischen Anonymität und Identifizierung abgedeckt werden können. Dies alles soll zum Erhalt der Privatsphäre und Kontrolle der eigenen Daten führen. Dazu muss ein Identitäts-Manager Funktionen zum Erstellen, Speichern, Verändern und Zugreifen auf digitale Identitäten bereitstellen. Es haben dabei jedoch immer die Absichten und Interessen des Nutzers im Vordergrund zu stehen. [KW01]

1	Schutz der persönlichen Daten
2	Kontrolle der Kommunikation durch den Benutzer
3	Vermeidung der Fragmentierung von Identitätsdaten
4	Steigerung der Qualität und Aktualität der angebotenen Daten
5	Offenlegung der verbreiteten Daten
6	Authentifizierung ermöglichen
7	Pseudonymisierung und damit Anonymität ermöglichen
8	Verhinderung des Identitäts-Diebstahls

Tabelle 3.1: Ziele des Identitäts-Managements

3.3 Bestehende Ansätze

In diesem Abschnitt werden zur Zeit gängige Identitäts-Infrastrukturen vorgestellt und miteinander verglichen. Dabei wird bei der Vorstellung der Architekturen an den gegebenen Stellen auf die in Abschnitt 3.2 gemachten Erwartungen an ein Identitäts-Management eingegangen. Der abschließende Vergleich der Ansätze stellt ihre Unterschiede und Gemeinsamkeiten heraus.

3.3.1 .NET Passport

Die Identitätsplattform .NET Passport wurde von der Firma Microsoft entwickelt und im Jahr 1999 in Betrieb genommen. Die primären Ziele dieses Ansatzes sind, die Benutzung des Internets und online Einkäufe zu vereinfachen und zu beschleunigen. [Mic02a] Dazu wurden Dienste zur Benutzer-Authentifizierung und zur Übertragung von ausgewählten Profildaten entwickelt.

Single Sign In (SSI)

Der in .NET Passport implementierte Dienst zur Authentifizierung von Benutzern bedient sich lediglich standardisierten Web-Techniken. Die benötigten Daten – Benutzername und Passwort – werden dabei zentral auf einem Server verarbeitet und verwaltet. [KW02]

Um den Single Sign-In Dienst nutzen zu können, muss der Benutzer sich zunächst auf der Web-Seite von .NET Passport mit seiner E-Mail-Adresse als Benutzernamen und einer frei wählbaren Zeichenkette als Passwort registrieren. Beim Registrierungsprozess kann der Benutzer pauschal entscheiden, ob seine E-Mail-Adresse den .NET Passport einsetzenden Inhalteanbietern nach erfolgter Authentifizierung zur Verfügung stehen soll. Zusätzlich kann der Nutzer neun vorgeschlagene Profildaten wie Namen, Geburtsdatum oder Geschlecht eingeben. Auch deren Weitergabe wird

pauschal durch den Nutzer autorisiert oder verweigert. In jedem Fall wird jedoch eine „Personal User-ID“ (PUID) an den Web-Server des Inhalteanbieters übertragen, mit der der Nutzer eindeutig identifiziert werden kann. Es gibt jedoch zwei Szenarien, in denen unabhängig von der Benutzereinstellung Profilinformationen weitergegeben werden [Mic02b, KW02]:

1. Wenn die Registrierung bei .NET Passport über eine Partnerseite vermittelt wurde, so erhält diese die eingegebenen Informationen.

2. Wenn die Partnerseite als Teil des .NET Passport-Kontos eine E-Mail-Adresse bereitstellt, erhält diese Seite die Profildaten bei jeder Anmeldung.

Das .NET Passport Modell umfasst drei Parteien. Erstens den Benutzer, der sich mittels seines Browsers durch das World-Wide-Web bewegt, zweitens den Anbieter einer Seite im Internet und drittens den zentralen Server mit den Login-Daten. Die Authentifizierung des Nutzers (vergl. Abbildung 3.4) beginnt mit dem Aufruf der Seite eines Inhalteanbieters, der .NET Passport als Login-Mechanismus verwendet (1.). Dieser Server beantwortet die Anfrage des Nutzers jedoch nicht direkt, sondern veranlasst mittels eines Redirects (2.) den Browser des Nutzers die Login-Seite des Authentifizierungs-Dienstes zu laden (3.). Dabei werden die Rückkehradresse und ein Seiten-Bezeichner im Header mitübertragen. Der Login-Server fragt dann den Benutzernamen und das Passwort verschlüsselt ab und verifiziert diese (4. und 5.). Bei erfolgreicher Authentifizierung wird der Browser durch einen erneuten Redirect (6.) zurück zur originären Web-Seite geleitet (7.). Dabei wird die PUID des Benutzers übertragen, so dass der Inhalteanbieter den Nutzer in seinem Datenbestand identifizieren kann. Zum Abschluss wird die Login-Kennung in einem Cookie beim Benutzer gespeichert (8.). Dies wird benötigt, um den Nutzer während der weiteren Navigation wieder identifizieren zu können. [KR00, KW02]

Express Purchase Service

Der Express Purchase Service ist ein optionaler Dienst, der den Benutzer beim Einkauf im Internet unterstützt. Dazu legt sich dieser in seinem .NET Passport-Profil eine „elektronische Brieftasche“ an, in der Zahlungs- und Lieferdaten zusammengefasst werden. Die eingegebenen Daten werden verschlüsselt gespeichert und stehen für eine automatisierte Kaufabwicklung zur Verfügung. Der Ablauf des Protokolls ist dem der Authentifizierung mittels SSI sehr ähnlich. Es wird lediglich vorausgesetzt, dass der Nutzer sich zuvor eingeloggt hat. Anstelle der Aufforderung zur Eingabe des Benutzernamens und des Passwortes können die Daten ausgewählt werden, die an den Shop-Betreiber gesendet werden dürfen. Diese werden dann im Redirect übertragen, wobei eine verschlüsselte Verbindung verwendet wird. Die Semantik der übertragenen Felder wird durch die verwendete Electronic Commerce Modeling Language bestimmt. [Mic02b, KR00]

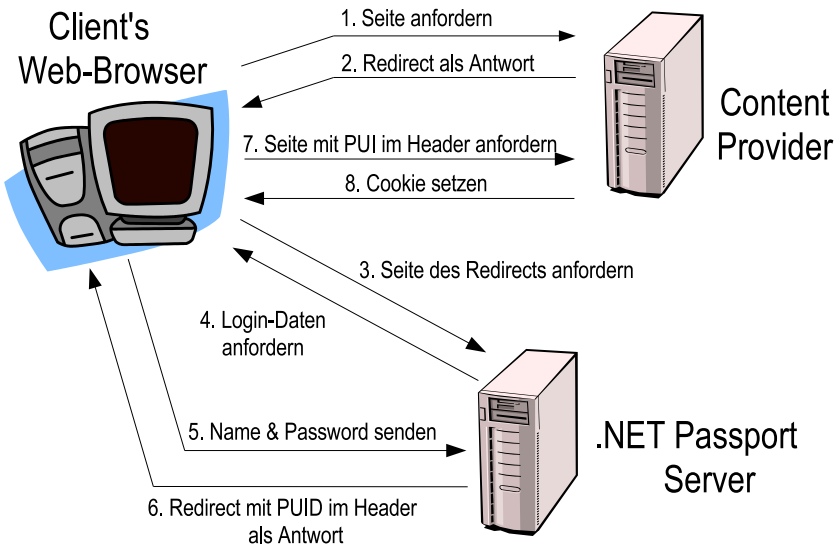

Abbildung 3.4: Single Sign-In mit .NET Passport nach [KR00]

3.3.2 Liberty Alliance Project

Das Liberty Alliance Projekt ist ein Zusammenschluss von verschiedenen Firmen und wissenschaftlichen Einrichtungen der Informations- und Kommunikationsbranche. Ziel des Projektes ist es, eine vernetzte Welt zu schaffen, in der es Individuen und Firmen möglich ist fast jede Transaktion ohne Verletzungen der Privatsphäre und Sicherheit des Einzelnen durchzuführen. Zur Umsetzung dieser Vision definiert die Liberty Alliance einen offenen Standard zur Repräsentation von Netzwerk-Identitäten. Außerdem werden Verfahren zur Authentifizierung der Nutzer und zur Zugriffskontrolle der Profilinformationen bereitgestellt. [HW03, Koc02]

Föderierte Netzwerk-Identität

Grundlage der Architektur des Projektes ist die Netzwerk-Identität. Diese ist definiert als *„die globale Menge der Attribute, die aus den verschiedenen Accounts des Individuums zusammengefasst ist“* [HW03, S. 7] Das bedeutet also, dass die personenbezogenen Informationen des Profils nicht zentral auf einem Server gespeichert werden, sondern bei den einzelnen Dienstleistern verbleiben. Um einen gewissen Grad an Datenschutz zu gewährleisten, bietet die Architektur der Liberty Alliance die Möglichkeit, dass sich Anbieter zu Gruppen zusammenschließen, in denen man sich untereinander vertraut (circle of trust). Innerhalb eines solchen Kreises kann der Nutzer seine sonst isolierten Accounts zu einer föderierten Netzwerk-Identität verbinden. Damit ist es dem Benutzer möglich nahtlos mit allen Teilnehmern der Gruppe in geschäftliche Beziehung zu treten. [KW02, HW03]

Beispielhaft könnte man die teilnehmenden Firmen an einem Bonusprogramm einer Fluglinie – wie dem Miles & More Programm der Lufthansa – als eine solche

Vertrauensgruppe betrachten. Diese Firmen vertrauen sich auf Grund ihrer geschäftlichen Beziehungen untereinander und können von der Zusammenarbeit profitieren. Aber auch der Nutzer hat durch den Bonus oder andere Vergünstigungen, die er erhält, einen Mehrwert, so dass daraus seine Bereitschaft zur Verknüpfung seiner bislang isolierten Accounts erwachsen könnte.

Das Liberty Alliance Projekt definiert in seinem Modell verschiedene Rollen der teilnehmenden Parteien. Die Zentrale Rolle kommt dabei dem Nutzer zu. Dieser tritt in Kommunikation mit Identitäts- und Diensteanbietern. Dabei haben die Identitätsanbieter die Aufgabe als zentrale Anlaufstelle zu fungieren, die den Status des Benutzers und die föderierte Netzwerk-Identität überwacht und verwaltet. Die Dienste-Anbieter offerieren dem Benutzer webbasierte Angebote. Dabei nutzen sie die Identitäts-Infrastruktur innerhalb der vertrauenswürdigen Gruppe. [HW03]

In obigen Beispiel könnte die Fluglinie der Identitätsanbieter sein und die angeschlossenen Unternehmen die Dienste-Anbieter. Dabei ist natürlich nicht ausgeschlossen, dass die Fluglinie auch ein Angebot für den Nutzer bereitstellt.

Liberty Architektur

Die Architektur des Liberty Alliance Projektes (vergl. Abbildung 3.5) basiert wie .NET Passport auf den heute verfügbaren Web-Technologien. Sie ist dabei in drei architektonische Grundkomponenten aufgeteilt [HW03]:

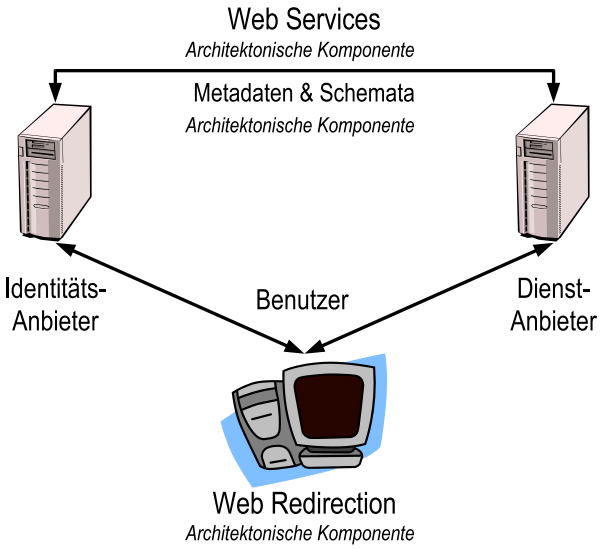

Abbildung 3.5: Liberty-Gesamtarchitektur nach [HW03, S. 19]

Web Redirection Dieser Mechanismus des HTTP-Protokolls bildet die Basis für das Single Sign-In und die föderierten Identitäten. Dabei wird der Web-Browser durch

Web Redirection	Handlung, die es den Dienstleistern in der Liberty-Architektur ermöglicht, heutige Mittel der Web-Technologie zur Kommunikation zu nutzen.
Web Services	Protokolle, die es den Einheiten ermöglichen direkt miteinander zu kommunizieren.
Metadata und Schemata	Eine Menge von allgemeingültigen Metadaten und Formaten, die es den Server-Betreibern erlauben, für Dienstanbieter typische und allgemeine Daten auszutauschen.

Tabelle 3.2: Architektonische Komponenten der Liberty Alliance nach [HW03, S.19]

einen Befehl automatisch auf eine beliebige andere Seite umgeleitet. In der zugrundeliegenden Nachricht können Daten eingebettet werden, die an den Server der neuen Seite mitübertragen werden. Koppelt man zwei Redirects hintereinander, so kann man mit Umweg über den Browser des Nutzers Daten austauschen. Dies wird in der Architektur der Liberty Alliance dazu verwendet, Informationen, wie zum Beispiel den Login-Status des Benutzers zu übertragen. Diese Methode ist notwendig, da meistens keine Möglichkeit besteht, einen gemeinsamen Speicher (Cookie) auf Clientseite einzurichten und darüber Nachrichten auszutauschen. Dabei bestehen die Restriktionen, die dies verhindern darin, dass die Web-Browser im Allgemeinen auf Grund ihrer Sicherheitseinstellungen nur die Server auf die Cookies zugreifen lassen, die diese auch gesetzt haben. [HW03]

Web Services Neben den Teilen der Protokolle, die über den Redirect-Mechanismus ablaufen, gibt es auch Teile, die direkt zwischen Anbietern übertragen werden. Diese dem Remote Procedure Call (RPC) ähnlichen Nachrichten werden dabei mit Hilfe des Simple Object Access Protocols (SOAP) ausgetauscht. SOAP basiert dabei auf XML und HTTP um seine Kommunikation durchzuführen und passt somit in die Web-Umbegung der Liberty Architektur. [HW03]

Metadaten und Schemata Unter diesen Begriffen sind eine Reihe von Informationen und Formaten zusammengefasst. Mit ihrer Hilfe können Identitäts- und Dienste-Anbieter Daten austauschen. Dabei gibt es Nachrichten mit definierten Inhalten und Nachrichten zum Übermitteln beliebiger Inhalte. Es werden zum Beispiel Attribute wie der mit dem Benutzer gekoppelte Bezeichner, Angaben zum Authentifizierungs-Verfahren oder andere Metadaten, die Dienstleister zuvor vereinbart haben ausgetauscht. [HW03]

Single Sign In Szenario

Das Single Sign In, dass die Liberty Alliance anbietet (siehe Abbildung 3.6), verläuft ähnlich dem Verfahren, welches durch .NET Passport bereitgestellt wird (vergl. Abschnitt 3.3.1). Zunächst läd der Nutzer die Seite des Anbieters, mit dem er in Kontakt treten möchte. Auf dieser Seite wählt er mittels eines Verweises das Log-In über den Identitäts-Anbieter der vertrauenswürdigen Gruppe (1.). Der Server des Dienste-Anbieters beantwortet die Anfrage nicht direkt, sondern veranlasst den Browser des Nutzers durch einen Redirect (2./3.) das Login über den zentralen Server der Gruppe vorzunehmen (4.). In der Redirect-Nachricht wird dabei die Rücksprungadresse auf die Seite des Dienstleisters mit übertragen. Nach einem erfolgreichem Log-In wird der Browser des Nutzers mit Hilfe eines weiteren Redirects zurück zum Dienste-Anbieter gelenkt. Dabei sind in der Redirect-Nachricht Angaben zum Benutzer eingebettet (5./6.). Dieser Bezeichner versetzt den Dienstleister nun in die Lage, gezielt Informationen über den Nutzer beim Identitätsanbieter zu erfragen (7.).

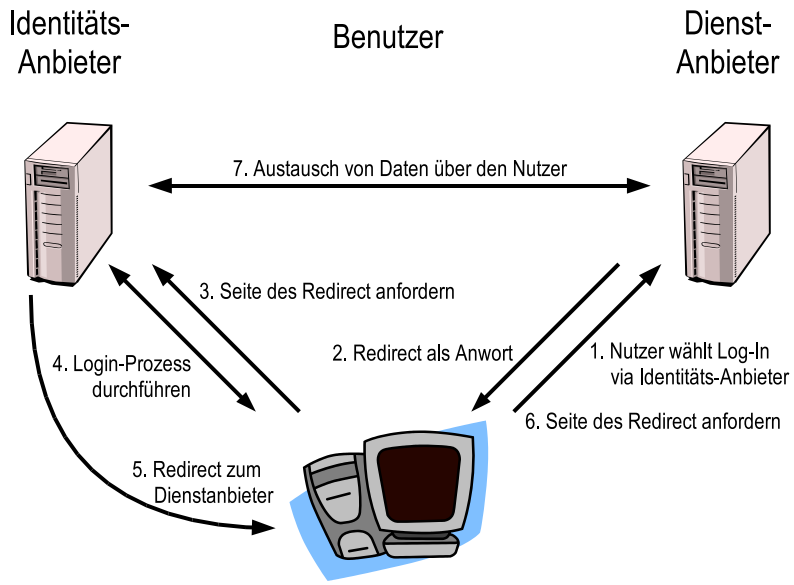

Abbildung 3.6: Single Sign-In mit der Liberty Architektur nach [HW03, S. 34]

3.3.3 eXtensible Name Service

Der eXtensible Name Service (XNS) ist ein auf XML basierender Web-Identitäts-Dienst. Er bietet ein Konzept einer globalen Identität und ein Beziehungs-Management. [XNS02c] Dabei umfasst die XNS-Plattform die drei Hauptkomponenten „Globales Identitäts-Management“, „Kontrolle über Privatsphäre und Sicherheit“ und „Automatischen Datenaustausch und Synchronisation“. [One03]

Globales Identitäts-Management

Im XNS Modell ist die Web-Identität ein logisches Konstrukt, dass verteilte Identitäts-Daten logisch organisiert und miteinander verknüpft. Dabei orientiert sich der Ansatz von XNS am Konzept des World-Wide-Webs (vergl. Abbildung 3.7). Es wird angenommen, dass Identitäts-Daten eines Nutzers, analog zu den Dateien im WWW, physikalisch auf mehrere Servern verteilt sein können. Diese Daten sollen durch eine einheitliche Repräsentation und durch Verknüpfung zu einer logischen Einheit verbunden werden. Damit entsteht als „Globale Identität“ ein Identitäts-Netz, analog zum Hypertext, der ein Netz aus Dokumenten darstellt. Um verschiedenartige Identitäts-Informationen miteinander zu einer Einheit zu vereinigen ist es nötig, diese anwendungsunabhängig zu repräsentieren. Dazu werden die Identitäts-Dokumente in XNS in der Sprache XML beschrieben. Die innere Struktur dieser Dokumente stellt dabei einen Baum von Attributen dar. Dabei wird von XNS bezüglich der Attribute lediglich verlangt, dass sie durch ein XML-Schema definiert sind. [XNS02a, XNS02c, XNS02b]

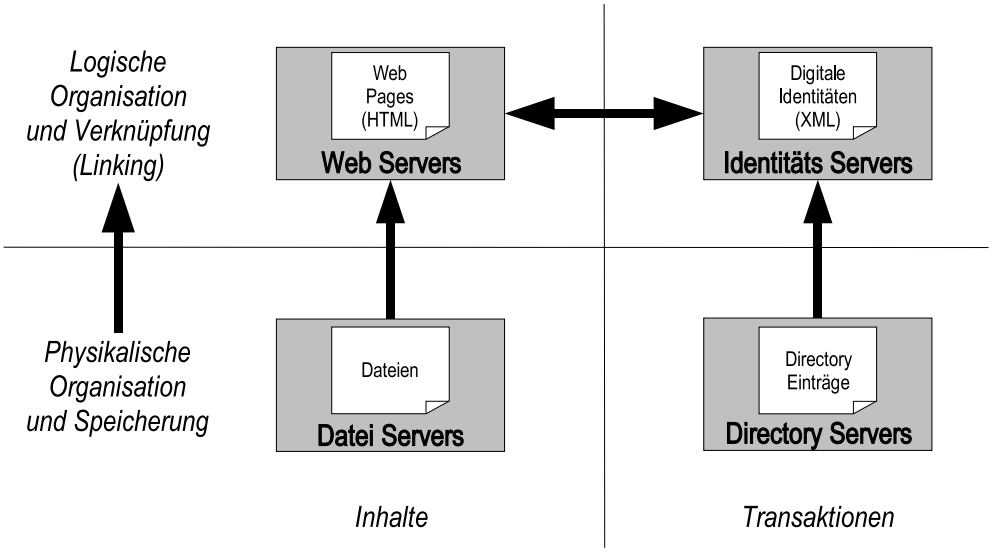

Abbildung 3.7: Analogie von Identitäts-Netz und WWW nach [XNS02b, S. 7]

Ein weiterer Teil des Management-Konzepts ist ein eigenes Adressierungs-Modell. Dieses dient der Persistenz der logischen Identität, auch dann, wenn Änderungen in deren Komponenten eintreten. Dazu wird eine weitere Abstraktionsschicht in das Modell eines Netzwerk-Namens-Dienstes eingefügt (vergl. Abbildung 3.8). Zu diesem Zweck wird jeder globalen XNS-Identität zur eindeutigen und dauerhaften Identifizierung ein einheitlicher Bezeichner gegeben. Mit Hilfe dieser Bezeichner kann jeder Teil einer Identität durch ein Uniform Resource Name (URN) eindeutig referenziert werden. Dieser zusätzlich Schritt ist notwendig, da sich die semantischen Namen von Identitäten oder Attributen während deren Lebenszeit verändern können. Ist also einmal die Auflösung des Namens in den persistenten Bezeichner vorgenommen,

kann sich der Name ändern, ohne dass der Inhalt unter dem alten Namen nicht mehr zugreifbar ist. [XNS02a, XNS02b]

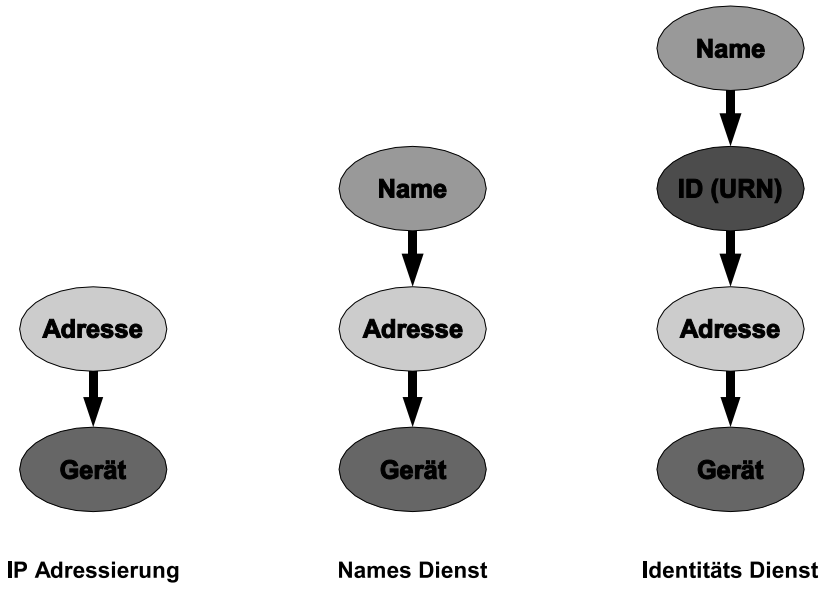

Abbildung 3.8: Von Namens- zu Identitätsauflösung nach [XNS02a, S. 23]

So könnte zum Beispiel in einer elektronischen Visitenkarte der Name des Attributs der privaten Anschrift von der deutschen Bezeichnung in die englische geändert werden. Alle diejenigen, die die Namensauflösung bereits betrieben haben, können trotzdem auch weiterhin auf den Inhalt des Attributs zugreifen (vergl. Abbildung 3.9).

Kontrolle über Privatsphäre und Sicherheit

Zur Kontrolle des Austausches von Attributen durch Verknüpfungen sind jedem Link sogenannte Verträg zugeordnet. In einem solchen Vertrag sind die Rechte und Regeln zur Zugriffskontrolle und zum Schutz der Privatsphäre festgelegt. So können zum Beispiel Lese- und Schreibrechte eingeräumt oder verweigert werden. [XNS02a]

Daneben gibt es in der XNS-Architektur weitere Dienste zum Erlangen von Sicherheit und Vertrauen. So gibt es einen Zertifizierungsdienst, der mit Hilfe digitaler Schlüssel Attribute oder Attributgruppen elektronisch unterschreibt. Dazu gehört eine Schlüsselverwaltung, mit deren Hilfe die öffentlichen Schlüssel zugänglich gemacht werden können. Als Basis für einen Single Sign In und zur Session-Verwaltung gibt es einen Dienst zur Authentifizierung. Dieser Dienst überprüft, ob ein Benutzer derjenige ist, der die Identität verwaltet. [XNS02d]

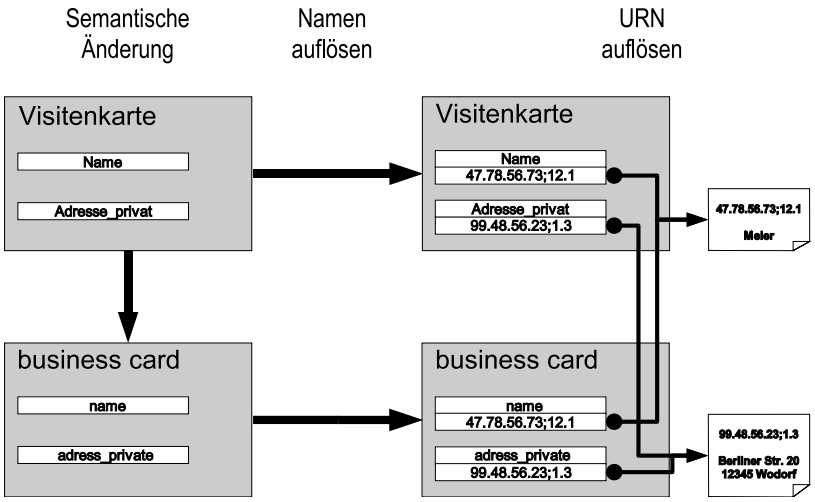

Abbildung 3.9: Beispiel zur Identitäts-Adressierung in XNS

Automatischer Datenaustausch und Synchronisation

Neben den Regeln und Rechten zur Kontrolle der Privatsphäre und Sicherheit können die Links auch speichern, welche Information mit wem ausgetauscht wurde, so dass bei Veränderungen ein automatischer Abgleich möglich ist. Auf diese Weise können zum Beispiel diejenigen, die zertifizierte Attribute erhalten haben, über die Rücknahme des Zertifikats unterrichtet werden. [One03, XNS02d]

3.3.4 Vergleich der Varianten

Vergleicht man die zuvor genannten Identitäts-Infrastrukturen, so kann man erkennen, dass das Konzept der Liberty Alliance und das von .NET Passport einen ähnlichen Fokus haben (vergl. Tabelle 3.3). Beide konzentrieren sich auf das World-Wide-Web und nutzen dessen Infrastruktur. XNS hingegen versteht sich als eine Erweiterung der dem Internet zugrundeliegenden Konzepte. Deshalb kann es in einer größeren Menge von Anwendungen Einsatz finden. In Kommunikationsnetzen, die nicht auf dieser Technologie basieren, ist der Einsatz hingegen schwierig. Topologisch setzen die Liberty Alliance und .NET Passport zentrale Server zur Koordination und zum Ausführen von Aufgaben, wie zum Beispiel Authentifizierung, ein. Dagegen sind die dezentral verteilten Identitäts-Server im Modell des eXtensibe Name Service gleichberechtigt. In diesem Punkt hat XNS einen Vorteil gegenüber den beiden anderen Ansätzen, da das System bei Ausfall eines Servers nur partiell ausfällt. Bei den Konzepten mit zentralem Server hingegen führt ein Ausfall zum Ausfall des gesamten Systems. Betrachtet man die Datenhaltung der drei Systeme, so fasst der Ansatz von .NET Passport alle Profilinformationen zentral an einer Stelle zusammen. Dies hat den Vorteil gegenüber der verteilten Speicherung, die die beiden anderen Konzepte erlauben, dass eine Fragmentierung der Identitätsdaten vermieden wird (vergl.

		Liberty Alliance	.NET Passport	XNS
Fokussierung auf	Universalität			◯
	World-Wide-Web	X	X	
Server	zentral	◯	X	
	dezentral			X
Datenhaltung	zentral		X	
	dezentral	X		X
Ort der Datenhaltung	beim Benutzer			
	beim Anbieter	X	X	X
Datenschutz	pauschal	X	X	
	fein gegliedert			X
Datenübermittlung	offengelegt			◯
	verborgen	X	X	

X := Erfüllt das Attribut
◯ := Erfüllt das Attribut teilweise

Tabelle 3.3: Eigenschaften der verschiedenen Ansätze

Abschnitt 3.2). Beim Ort der Datenhaltung stimmen die drei Konzepte überein. Sie speichern die Daten des Profils jeweils auf der Seite der Anbieter und damit außerhalb der direkten Kontrolle des Nutzers. Die Regeln und Rechte, mit denen der Benutzer seine Daten vor unautorisiertem Zugriff schützen kann, setzen bei der Liberty Alliance und bei .NET Passport auf einer hohen Abstraktionsebene an. Das Microsoft Konzept erlaubt es lediglich pauschal anzugeben, ob die Attribute an Dienste-Anbieter übertragen werden dürfen oder nicht. Im Liberty Projekt kann man diese Angaben für föderierte Gruppen und jeden Account einzeln entscheiden. Bei XNS hingegen kann für jedes einzelne Attribut entschieden werden, wer welche Zugriffsrechte bekommt. Damit ist dieser Ansatz im Bereich des Datenschutzes wesentlich mächtiger als die beiden anderen Architekturen. Auch bei der Offenlegung der an die einzelnen Anbieter übertragenen Daten bietet XNS dem Nutzer die meisten Informationen der drei Ansätze. Dadurch, dass für jedes Attribut und jeden Kommunikationspartner eigene Regeln vorhanden sind, kann indirekt auf die Verbreitung der Profilinformationen geschlossen werden. Die Konzepte von .NET Passport und der Liberty Alliance hingegen bieten dem Nutzer keine Möglichkeit zu erfahren, wer wann welche Daten bekommen hat.

Kapitel 4

Entwurf einer Identitäts-Infrastruktur

Dieser Teil der Arbeit setzt die in Kapitel 3 erlangten Erkenntnisse in einen konkreten Designentwurf um. Dazu wird zunächst eine Abwägung zwischen möglichen Darstellungsvarianten der Identität vorgenommen. Daraus wird dann das onefC-Identitäts-Modell abgeleitet. Danach wird der Entwurf des in der Architektur verwendete Identitäts-Managers mit seinen einzelnen Dienstkomponenten vorgestellt. Abschließend wird ein Protokoll-Ansatz zum späteren Test des Prototyps eingeführt.

4.1 Identitäts-Rahmenmodelle

Wie in den Abschnitten 3.1.1 und 3.1.2 beschrieben, hat die Identität im zwischenmenschlichen Bereich verschiedene Aufgaben. Ein Identitätskonzept sollte nach Möglichkeit viele dieser Punkte abbilden. Dazu gehört zum einen die eindeutige und konsistente Identifizierung und Zuordnung der Identität über die Zeit und unterschiedliche Kontexte hinweg. Zum anderen muss eine Identitätsabbildung die Auswahl und Anpassung der präsentierten Attribute entsprechend der eingenommenen Rolle und Situation erlauben. Außerdem sollten die Attribute keiner inhaltlichen Einschränkung unterliegen. Dies ist wichtig, um das Konzept nicht auf eine kleine Menge von Anwendungen einzugrenzen. Es ist wünschenswert, die Möglichkeit zu haben, Attribute, die ihren Ursprung bei einer fremden Identität haben, in die eigene zu übernehmen. Dabei sollte die Herkunft dokumentiert sein, damit diese zum Beispiel als Referenz oder Leumund dienen kann. Auch die Abbildung von Sozialen Identitäten, also das sich zu einer Gruppe Zugehörigfühlen, sollte durch das Konzept mit abgedeckt sein. Eine aus eher praktischen Erwägungen gestellte Anforderung ist die Möglichkeit der Wiederverwendung von einmal gespeicherten Attributen in unterschiedlichen Kontexten. Diese Punkte sind in der Tabelle 4.1 nochmals zusammengefasst.

An dieser Stelle sollen zunächst einige Begriffe näher betrachtet werden, so wie sie in diesem Konzept verwendet werden:

1	Eindeutige und konsistente Identifizierung über die Zeit
2	Auswahl der im Kontext präsentierten Attribute
3	Inhaltliche Anpassung der Attribute je nach Kontext
4	Keine Einschränkungen bei den Inhalten der Attribute
5	Aufnahme von Attributen fremden Ursprungs
6	Möglichkeit der Abbildung von Sozialen Identitäten
7	Möglichkeit der Wiederverwendung von Attributen anderer Kontexte

Tabelle 4.1: Anforderungen an eine Identitätsabbildung

1. **Kontext**
   Unter einem Kontext versteht man einen gemeinsamen Zusammenhang, Hintergrund oder ein Umfeld. [BRO89a] Bezogen auf das Identitäts-Modell, soll ein Kontext ein allgemeiner Erfahrungshorizont sein, in dem ein Teil der Identität präsentiert wird. Er kann sich somit sowohl auf eine eingenommene Rolle als auch auf die zugrundeliegende Situation beziehen. Damit kann der Kontext als Schlüssel für die Auswahl der in der aktiven Identität enthaltenen Attribute und deren Inhalte dienen. Tritt zum Beispiel eine Person als Softwareentwickler auf, so könnte sie zur Beschreibung ihrer gerade eingenommenen Rolle einen international normierten Kontext verwenden. Durch eine Normung wäre der Kontext sprachunabhängig und könnte global eingesetzt und verstanden werden.

2. **Soziale Identität**
   Soziale Identitäten entwickeln sich wie in Abschnitt 15 beschrieben in Gruppen. Jedes sich der Gruppe zugehörigfühlende Individuum trägt mit seiner Identität seinen Teil zur virtuellen Gruppen-Identität bei. Das bedeutet, dass die Soziale Identität keine Sonderform einer personalen Identität ist. Vielmehr ist lediglich das Gefühl der Zugehörigkeit, also die Ingroup-Beziehung, ein Attribut in den Identitäten der Gruppenmitglieder. Die Soziale Identität ergibt sich dann aus der Verknüpfung aller Identitäten der Gruppe. Dies ist in Abbildung 4.1 veranschaulicht. In der Grafik stehen $A_{1...5}$ für beliebige Attribute der Identität. A_{SID} ist das Attribut mit dem die Beziehung zur Gruppe hergestellt wird. Durch diese Verknüpfung werden jeweils die eigenen Attribute virtuell der Sozialen Identität hinzugefügt (gestrichelte Linie).

4.1.1 Eindeutige Identifizierung

Die erste Anforderung an die Abbildung der Identität ist ihre eindeutige Identifizierung über die Zeit und verschiedene Kontexte hinweg. Diese Forderung schränkt die Menge der möglichen Verfahren bereits ein. Methoden, die auf nur temporär gültigen

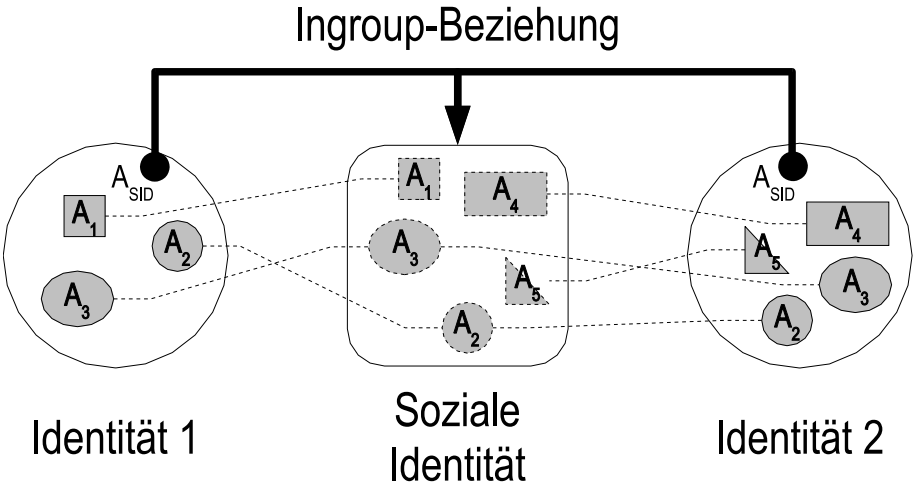

Abbildung 4.1: Verdeutlichung des Begriffs der Sozialen Identität

Werten basieren können keine Verwendung finden. So zum Beispiel die Identifizierung der Identität anhand ihrer Position im Hauptspeicher des Rechners. Die Wiederherstellung einer persistent gemachten Identität oder Änderungen in der Struktur führen im Allgemeinen zu einer Verlagerung des Objektes im Speicher des Computers. Damit wäre die eindeutige Wiedererkennung nicht mehr möglich. Weitere Einschränkungen ergeben sich, durch die Verwendung innerhalb von Netzwerken. Alle Methoden, die lediglich lokale Eindeutigkeit gewährleisten, können ebenfalls nicht angewendet werden. Dazu gehört zum Beispiel der Einsatz eines lokalen Zählers, der jeder Identität eine Nummer zuordnet. Ein solcher Zähler kann keine Überschneidungen der Bezeichner auf unterschiedlichen Rechnern verhindern. Dazu müsste es einen zentralen Zählerdienst geben, der die Bezeichner für alle Identitäten vergibt. Die Verwendung eines solchen Dienstes würde jedoch erzwingen, dass neue Identitäten nur dann angelegt werden können, wenn der Dienst erreichbar ist. Da jedoch Identitäten unabhängig von bestehenden Netzverbindungen erzeugt werden sollen, kann auch dieser Ansatz keine Verwendung finden. Es müssen also weltweit eindeutige Bezeichner verwendet werden, die von jedem selbst erzeugt werden können. Bezeichner, die diese Anforderung erfüllen sind zum Beispiel Universal Unique Identifier (vergl. Abschnitt 2.1.1) oder Globaly Unique Identifier. Diese Bezeichnerarten sind so aufgebaut, dass eine Überschneidung zwischen zwei Bezeichnern fast ausgeschlossen ist. Damit können sie als Basis für die eindeutige Identifizierung innerhalb des Identitätskonzepts verwendet werden.

4.1.2 Architektur

Die strukturelle Abbildung der Identität kann auf verschiedenste Weise vorgenommen werden. Zwei der möglichen Konstrukte sollen an dieser Stelle vorgestellt werden. Sie orientieren sich zum einen an den eher klassischen Identitäts-Konzepten und zum

anderen an dem moderneren Verständnis des psychologischen Identitätsbegriffs.

Identitätsabbildung mittels Schablonen

Das Schablonenkonzept zur Abbildung von Identitäten basiert auf zwei Datenspeichern (vergl. Abbildung 4.2). Der eine Speicher enthält die Profildaten und der andere die Schablonen. Der Benutzer muss seine Daten lediglich einmal in der Profildatendatei ablegen, um sie in seinen Teil-Identitäten verfügbar zu machen. Die Schablonen dienen dazu, die Attribute aus den Profildaten auszublenden, die nicht Teil der momentan aktiven Identität sind. Hierdurch wird die in der Identität präsentierte Attributmenge bezüglich des Kontextes eingeschränkt. Es ist also zuerst eine Abbildung des aktuellen Kontextes auf eine der Schablonen vorzunehmen, die dann mit den Profildaten zur eigentlichen Identität verknüpft wird (vergl. [JG00, JGM01]).

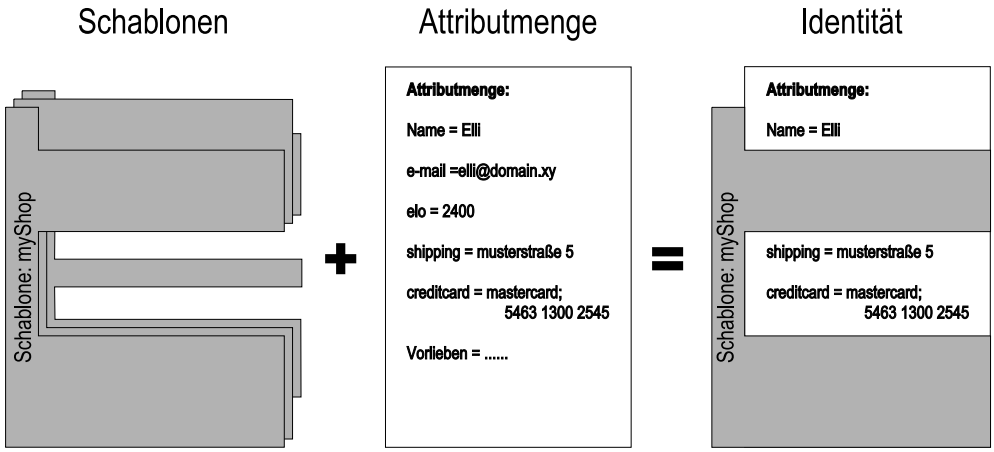

Abbildung 4.2: Schablonenkonzept zur Abbildung von Identitäten

Das Schablonenkonzept erfüllt die Anforderungen der Auswahl der in einem Kontext repräsentierten Attribute und deren Wiederverwendung in unterschiedlichen Szenarien sehr gut. Jedoch ist eine inhaltliche Anpassung der Attribute je nach Kontext nicht direkt möglich. Lösungen dieses Problems, führen im Allgemeinen zu Inkonsistenzen im Konzept. Entweder muss man kontextbezogene Informationen mit in der Profildatei abspeichern oder Profildaten mit in die Schablonen schreiben. Konkreter gesagt könnte man Attribute mehrmals – aber mit verschiedenem Inhalt – in die Profildatei aufnehmen und die unterschiedlichen Varianten den Schablonen zuordnen. Anhand der verwendeten Schablone würde dann der Inhalt des Attributes bestimmt. Ein anderer Weg wäre die Inhalte der veränderten Attribute direkt in den Schablonen abzulegen und bei der Verknüpfung mit den Profildaten die Inhalte der Attribute entsprechend zu ersetzen.

Auf Grund der Integration aller Attribute und damit auch aller Teil-Identitäten in einer Profildatei kann dieses Konzept dem klassischen Verständnis des Identitäts-Begriffs zugeordnet werden (vergl. Abschnitt 3.1.2). Das Schablonenkonzept stellt mit

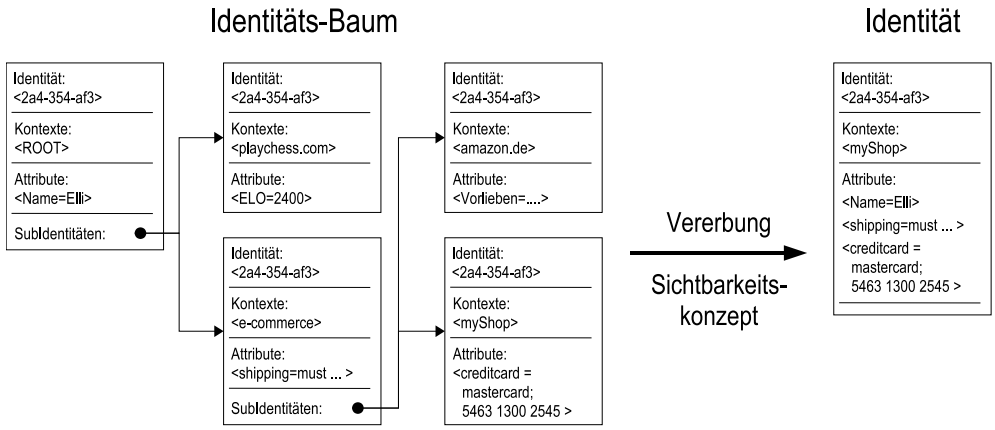

Abbildung 4.3: Baum-Konzept zur Abbildung von Identitäten

der Profildatei einen zentralen Ort zur Verfügung, der für Konsistenz und Kontinuität steht.

Baum-Darstellung

Das innerhalb des Projektes onefC entwickelte Konzept der Darstellung der Identität als Baum ist durch die neueren Konzepte des Identitäts-Begriffs inspiriert (vergl. Abschnitt 3.1.2). Die Aufteilung der Identität in verschiedene Knoten soll das „Identitäts-Patchwork“ abbilden (vergl. Abbildung 4.3). Jeder der Identitäts-Knoten steht für eine Teil-Identität, die in einem oder mehreren Kontexten aktiv ist. Jeder Knoten enthält neben den Kontexten auch die Attribute, die in der durch sie repräsentierten Identität enthalten sein sollen. Um innerhalb der Baumstruktur die selben Attribute in verschiedenen Teil-Identitäten verwenden zu können, kann ein Vererbungskonzept eingeführt werden. Dieses sagt aus, dass alle Attribute, die sich in einem Elternknoten befinden auch Teil des Kindes werden. Durch rekursive Anwendung werden Attribute über mehrere Hierarchieebenen vererbt. Auch spezielle Verweisattribute, die die Attribute eines Knotens virtuell in einen anderen einbinden, sind für eine Wiederverwendung von Attributen denkbar. Durch die Einführung eines Sichtbarkeitskonzeptes, ähnlich wie in Programmiersprachen, kann auch die Abstimmung von Attributinhalten auf den jeweiligen Kontext erfolgen. Die Inhalte von geerbten Attributen wird dabei lokal im entsprechenden Knoten überschrieben. Dies hat eine Änderung des Inhaltes in dem Knoten selbst und seinen Kindern zur Folge. Alle anderen Knoten, die dieses Attribut besitzen sind von der Anpassung hingegen nicht betroffen.

4.2 Attribut-Modell

Das in der onefC-Identität verwendete Attribut-Modell soll so gestaltet sein, dass ein möglichst breites Spektrum an Anwendungen diese Identitäten nutzen kann. Eine

solche Abbildung sollte demnach kaum inhaltliche Einschränkungen machen (vergl. Tabelle 4.1). Dies bedeutet, eine ausgewählte Menge von vorgegebenen und semantisch festgelegten Attributen reicht nicht aus. Vielmehr könnte man versuchen, ein generisches Attribut-Modell zu konzipieren, mit dem eigene anwendungsspezifische Attribute definiert werden können. Ein anderer Weg ist es lediglich ein Container-Modell zu definieren, dass die Attribute der einzelnen Anwendungen unabhängig von ihrer semantischen Bedeutung aufnimmt. Ein solcher Container könnte zusätzlich Metadaten enthalten, die nähere Angaben zum Inhalt darstellen. Dies erleichtert den Umgang auch mit anwendungsfremden Inhalten. Außerdem könnten leicht zusätzliche Dienste bereitgestellt werden, die zwischen verschiedenen Repräsentationen eines Inhalts übersetzen.

Das Projekt onefC verwendet für die Profilattribute ein solches Container-Modell. Daneben gibt es aber auch semantisch festgelegte Attribute. So wird zum Beispiel die Zugehörigkeit zu einer Sozialen Identität als ein spezielles Attribut abgebildet. Dies schränkt das Einsatzspektrum jedoch nicht ein. Vielmehr werden die semantischen Ausdrucksmöglichkeiten der Identität an sich erweitert. Da die onefC-Identität in einem verteilten System Verwendung finden soll, ist dennoch eine Einschränkung nötig. Es muss von den Attributinhalten verlangt werden, dass sie in eine Repräsentation gebracht werden können, die es erlaubt die Attribute über ein Netzwerk zu versenden. Um möglichst unabhängig von den verwendeten Netzstrukturen und Methoden der Übertragung zu sein, ist die Abbildung auf die Extensible Markup Language (XML) gewählt worden. Ein weiterer Vorteil genau dieser Repräsentationsform ist, dass das Einsatzspektrum dadurch nur minimal eingeengt wird.

4.3 Das onefC-Identitäts-Modell

Das onefC-Identitäts-Modell verwendet als Rahmen eine Baumstruktur und für die enthaltenen Profilattribute ein Container-Modell. Daneben gibt es noch ein Attribut, welches die personale Identität zur sozialen erweitert. Der detaillierte Aufbau ist in Abbildung 4.4 als UML-Klassendiagramm dargestellt.

Im Zentrum steht der Identitäts-Knoten (`Identity`). Dieser ist mit einem eindeutigen Bezeichner (`Identifier`) versehen. Die Bezeichner aller Knoten eines Identitäts-Baumes haben den identischen Wert. Damit ist die Möglichkeit gegeben, die einzelnen Teil-Identitäten der verschiedenen Kontexte miteinander zu verknüpfen, auch wenn keine Identifizierung über die Attribute möglich ist. Die Forderung nach einer eindeutigen und über die Zeit konsistenten Identifizierung ist damit erfüllt. Wird jedoch der Bezeichner bewusst bei der Übermittlung einer Teil-Identität verändert, so kann Pseudonymität erlangt werden.

Der Identitätsknoten enthält neben dem Bezeichner auch eine Liste mit mindestens einem Kontext (`Context`), für die er gültig ist. Jeder dieser Kontexte wird selbst durch einen eindeutigen Bezeichner repräsentiert. Es wird für das Modell definiert, dass verschiedene Kontextinstanzen sich dann auf ein und den selben beziehen,

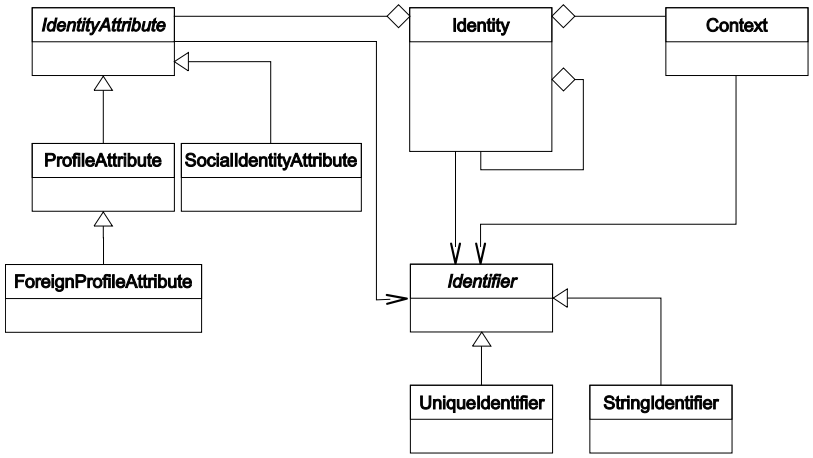

Abbildung 4.4: Klassendiagramm der onefC-Identität

wenn deren Bezeichner den gleichen Wert haben.

Jedes `Identity`-Objekt beinhaltet eine beliebige Menge an Attributen. Das Modell erlaubt es Attribute nur dann in eine Identität aufzunehmen, wenn sie von *`IdentityAttribute`* abgeleitet sind. Um Attribute innerhalb der Baum-Hierarchie überschreiben zu können, ist es notwendig, die einzelnen Attribute eindeutig miteinander in Beziehung zu setzten. Auch die Verwendung von Rechten und Regeln zur Kontrolle der Preisgabe der Identitäts-Attribute verlangt eine eindeutige Zuordnung. Aus diesen Gründen besitzen ebenfalls alle Attribute einen eindeutigen Bezeichner.

Zur Zeit definiert und verwendet das onefC-Modell drei Attribut-Typen. Die am häufigsten gebrauchten Typen sind das `ProfileAttribute` und seine Erweiterung als `ForeignProfileAttribute`. Diese definieren das Container-Modell für lokal- und fremderzeugte Profil-Attribute. Das `SocialIdentityAttribute` stellt hingegen den Bezug der Identität mit einer Gruppe her.

Alle Attribute besitzen ein optionales Merkmal, mit dem ihre Lebensdauer beschränkt werden kann. Dieser Eintrag ist mit `ttl` für „time-to-live“ bezeichnet und wird in Millisekunden seit dem 1. Januar 1970 00:00:00 GMT kodiert. Die Inhalte der Profil-Attribute, sind im onefC-Modell als Name/Wert Paare definiert und sollten eine XML-Repräsentation besitzen. Der Name ist dann gleich dem des Rootelements des Inhalts. Das zu Grunde liegende XML-Schema ist in dem Merkmal `schema` des Profil-Attributs gespeichert. Diese beiden Angaben können dazu genutzt werden, gezielt Attribute einer Identität abzufragen. Fremdprofil-Attribute besitzen als zusätzlichen Eintrag noch den Bezeichner der Identität, die es erzeugt hat. Attribute, mit denen man sich einer Gruppe zuordnet (`SocialIdentityAttribute`), können mit einem Alias versehen werden. Damit kann der Benutzer die Gruppe lokal umbenennen, um zum Beispiel einer mit einer UUID versehenen Gruppe einen verständlicheren Namen zu geben.

4.4 Der onefC-Identitäts-Manager

Dieser Abschnitt stellt die Umsetzung des Identitäts-Managers der onefC-Architektur vor. Dabei wird, bevor auf die einzelnen Komponenten eingegangen wird, ein konzeptioneller Überblick über deren Zusammenwirken gegeben.

4.4.1 Konzeptioneller Überblick

Der onefC-Identitäts-Manager soll Anwendungen als integrativer und infrastruktureller Dienst eine einheitliche Plattform zur Verwaltung von eigenen und fremden Identitäten bieten (vergl. Abbildung 4.5). Eine der Hauptaufgaben des Managers gegenüber den ihn nutzenden Anwendungen ist die Kapselung der Identitätskommunikation. Durch anwendungsübergreifende Integration aller identitätsrelevanten Daten, ermöglicht der onefC-Manager einen Datenaustausch ohne explizite Kenntnis der verschiedenen Applikationen untereinander. Außerdem wird durch seinen Einsatz eine Fragmentierung des persönlichen Profils vermieden.

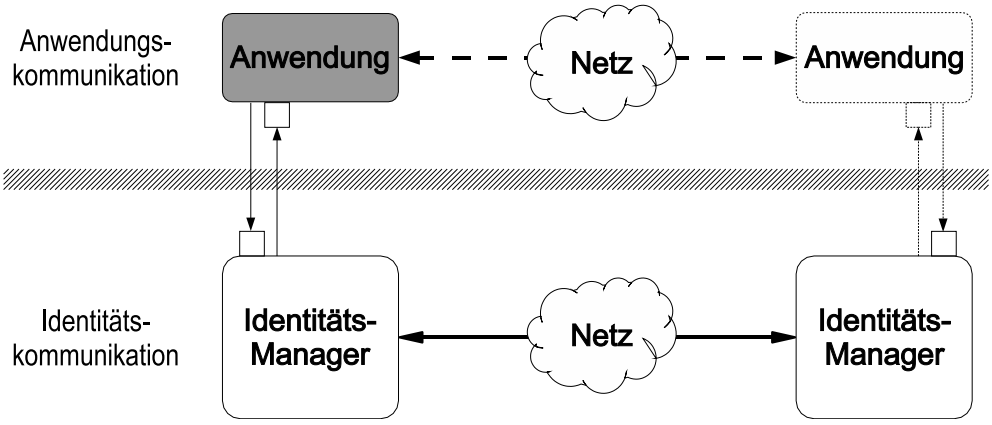

Abbildung 4.5: Identitäts-Management als infrastruktureller Dienst

Die Abbildung 4.6 zeigt, dass der onefC-Identitäts-Manager, um seine Leistung anbieten zu können, weitere Dienste in Anspruch nimmt. Jeder dieser Dienste ist als eigenständige Komponente konzipiert. Der Manager, die Services und die Anwendung kommunizieren mittels wohl definierter Schnittstellen. Dies und die Möglichkeit der dynamischen Integration neuer Komponenten, ist die Basis der Flexibilität und Adaptionsfähigkeit dieser Architektur. Die vom Manager genutzten Komponenten können in Dienste zum Erlangen von Sicherheit und Vertrauen, Netzwerkanbindung, Hilfsdienste und die Persistenz aufgeteilt werden.

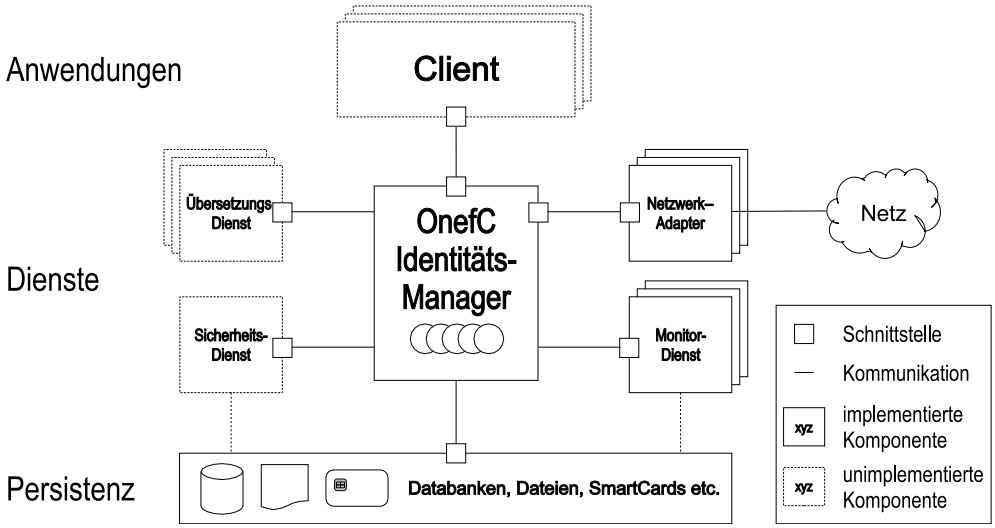

Abbildung 4.6: Komponenten des onefC-Identitäts-Managers

4.4.2 Die zentrale Management-Komponente

Leistungen gegenüber lokalen Anwendungen

Der Zugriff auf die Identitäts-Infrastruktur geschieht über die Schnittstelle der zentralen Management-Komponente. Anwendungen, die über den Eingang von Nachrichten oder die Aktualisierung der vom Identitäts-Manager verwalteten Fremdidentitäten informiert werden möchten, können sich bei ihm als Listener registrieren lassen. Der Grund für die Verwendung des Observer-Musters liegt in der asynchronen Kommunikationsbeziehung zwischen den Managern, die so auf die Anwendungsebene übertragen wird.

Die Aufgaben des Identität-Managers sind die Verwaltung der eigenen und fremden Identitäten. Darunter ist vor allem auch das Erzeugen der Identitäten die er enthält. Dabei werden die Identitäten des lokalen Nutzers direkt erzeugt. Die Fremdidentitäten hingegen basieren stets auf von anderen Managern übermittelten Daten. Um diese Daten zu erhalten, stellen die Anwendungen Anfragen an fremde Identitäts-Manager. Durch Auswahl eines der vom Manager verwalteten Network-Adapter, kann die anfragende Anwendung den Transport der Nachricht beeinflussen. Eine weitere wichtige Aufgabe der zentralen Komponente ist die Speicherung der lokalen Identitäten.

Zur Offenlegung der vom Manager ausgeführten Aktionen, können sich Monitoringdienste bei ihm registrierten. Diese werden dann über jegliche Kommunikation informiert.

Leistungen gegenüber fremden Identitäts-Managern

Die Manager kommunizieren untereinander zum Austausch von Identitätsdaten. Die Aufgabe, die der zentralen Komponente hierbei zukommt, ist die Erstellung einer Antwort auf eine eingegangene Nachricht. Dies geschieht in enger Abstimmung mit dem Sicherheitsdienst. Alle Attribute werden daraufhin überprüft, ob sie weitergegeben werden dürfen. Vor dem Versand wird zusätzlich abgefragt, ob die zu übertragende Identität zu pseudonymisieren ist. Dieses Verhalten stellt einen wesentlichen Teil der Abbildung der Selbstdarstellung dar.

4.4.3 Dienste für Sicherheit und Vertrauen

Eines der wesentlichen Ziele des Identitäts-Managements ist der Schutz der persönlichen Daten des Benutzers. Dazu zählt vor allem, dass der Identitätsinhaber die Weitergabe seiner Daten selbst kontrollieren kann (vergl. Abschnitt 3.2). Durch eine Protokollierung der Kommunikation, kann der Nutzer erkennen, ob die von ihm gemachten Weitergabe-Restriktionen auch wirklich eingehalten werden. Des weiteren wird durch ein solches Protokoll offengelegt, wer welche Daten von ihm besitzt. Die Selbstbestimmtheit der Datenweitergabe und die Offenlegung der verbreiteten Daten kann zu einer Steigerung des Vertrauens des Nutzers in den Manager führen. Dieses Vertrauen kann sich dann implizit auch auf die Anwendungen, die den Identitäts-Manager verwenden, übertragen und deren Akzeptanz steigern.

Die onefC-Manager-Architektur lagert die Aufgaben der Autorisierung der Datenweitergabe und das Protokollieren in zwei Komponenttypen aus. Dies ist zum einen der Sicherheitsdienst und zum anderen sind es die Monitordienste.

Sicherheitsdienste

Der Sicherheitsdienst (`SecurityService`) wird als eigenständige Komponente über eine vordefinierte Schnittstelle in die onefC-Architektur eingebunden. Jeder Identitäts-Manager steht mit einem Sicherheitsdienst in einer eis-zu-eins Beziehung. Dies bedeutet, dass immer nur eine Implementierung der `SecurityService`-Schnittstelle vom Manager verwendet wird. Diese Einschränkung ist gemacht worden, damit sichergestellt ist, dass der vom Benutzer gewählte und durch ihn kontrollierte Sicherheitsdienst auch Verwendung findet. Damit ist es Anwendungen nicht möglich, den vom Identitätsinhaber eingesetzten Schutz durch Einfügen eines eigenen Sicherheitsdienstes zu umgehen.

Dienste, die diese Schnittstelle implementieren, bieten den Identitäts-Managern drei sicherheitsrelevante Methoden an (vergl. Abbildung 4.7):

1. **Daten-Autorisation**
   Die zentrale Aufgabe einer `SecurityService`-Implementierung ist die Autorisation der Profildaten, die an einen Kommunikationspartner übermittelt werden dürfen. Um eine möglichst feine Granularität der Kontrolle des Nutzers

über seine Daten zu ermöglichen, ist die Autorisation auf Ebene der Attribute angesiedelt. Jedes Attribut wird bevor es an den Kommunikationspartner übermittelt wird an den Sicherheitsdienst übergeben. Dieser kann die Übertragung des Attributes dann auf drei Arten beeinflussen. Entweder stimmt er der Weitergabe zu oder er verweigert diese. Es ist ihm zusätzlich möglich, das Attribut zu verändern, bevor es der ausgehenden Nachricht hinzugefügt wird.

2. **Identitäts-Zuordnung**
   Eine weitere Aufgabe des Sicherheitsdienstes ist die Auswahl des zu verwendenen Identitätsbaumes, bei einer unspezifizierte Identitätsanfrage. Eine solche Nachricht, bei der der Fragende noch keine Information über den Kommunikationspartner hat, wird im allgemeinen die initiale Nachricht sein. Beim Eintreffen einer solchen Anfrage fragt der Identitäts-Manager über die Methode `makeChoice()` den Sicherheitsdienst nach dem Identitätsbaum, den er zur Beantwortung heranziehen soll. Zusätzlich übermittelt der Sicherheitsdienst im Rückgabeobjekt, ob der Manager die Identität unter einem Pseudonym präsentieren soll.

3. **Zugriffskontrolle**
   Zusätzlich stellt der Sicherheitsdienst noch eine Zugriffskontrolle mit der Methode `checkLogin()` zur Verfügung. Diese kann zum Beispiel verwendet werden, um nur Anwendungen Zugriff auf den Manager und damit auf die enthaltenen Identitäten zu gestatten, die im Besitz einer Berechtigung sind.

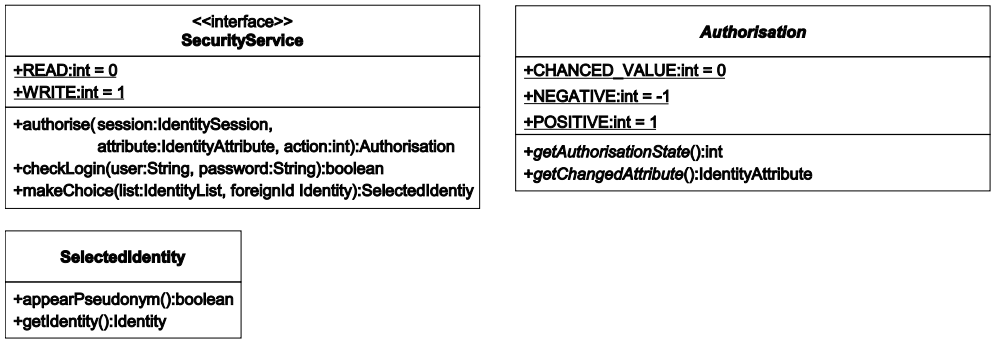

Abbildung 4.7: Klassen des Security-Pakets der onefC-Architektur

Monitoringdienste

Zur Erstellung von Kommunikationsprotokollen, können sich beim Identitäts-Manager Monitore registrieren lassen. Diese werden nach dem Observer-Muster über ein- und ausgehende Nachrichten informiert. Um sich beim Manager registrieren zu können, muss eine Komponente die `MonitoringService`-Schnittstelle implementieren (vergl.

Abbildung 4.8). Durch die Schnittstelle ist die Methode `notice()` definiert, über die der Manager die zu protokollierenden Daten an die Monitore übergibt. Um die Forderung der Offenlegung der verbreiteten Daten unabhängig von der Anwendungsebene zu gestalten, registriert der Manager immer selbst einen Monitor, der eine Protokolldatei erstellt.

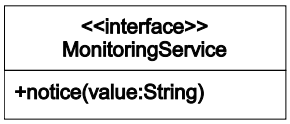

Abbildung 4.8: Schnittstelle der Monitoringdienste der onefC-Architektur

4.4.4 Netzwerkanbindung

Das Adapterkonzept

Ein Ziel des onefC-Identitäts-Managers ist es, Netzwerkunabhängigkeit zu erhalten. Um dieses erreichen zu können, wird für den Netzwerkanschluss ein Adapterkonzept verwendet.

In der Manager-Architektur ist hierzu eine Schnittstelle (`NetworkAdapter`) für die Netzwerkkommunikation definiert, mittels der Nachrichten versendet werden können (vergl. Abbildung 4.9). Eine Implementierung des Netzwerkadapters hat die Aufgabe, diese Kommunikationsschnittstelle an das von ihm verwendete Transportprotokoll anzupassen. Eingehende Nachrichten werden an Listener (Observer) gemeldet, die sich zuvor beim jeweiligen Adapter registriert haben. Dazu muss der „Beobachter“ die Schnittstelle `NetworkListener` implementieren. Um über die allgemeine Schnittstelle kommunizieren zu können, muss neben der Nachricht noch eine adapterspezifische Beschreibung der Verbindung (`ConnectionDescription`) angegeben werden. Die dort verwendete abstrakte Klasse der Netzwerkadresse (`NetworkAdress`) ist lediglich aus semantischen Gründen eingefügt. Sie hat keinerlei vordefinierte Funktionalität. Sie ist vielmehr auf das verwendete Protokoll abzustimmen.

Vorteile der asynchronen Kommunikation

Zum Austausch der Identitäten und Identitätsanfragen verwendet der onefC-Manager ein asynchrones Kommunikationsparadigma. Dies bedeutet, dass Nachrichten und deren Antworten nicht zeitlich aufeinander abgestimmt sind. Ein Grund hierfür ist, dass die Generierung von Antworten auch manuell beeinflusst sein kann. In einem solchen Fall kann keine Garantie über Beantwortungszeiten gegeben werden. Die Zuordnung der aus- und eingehenden Nachrichten geschieht über Referenzvariablen. Diese werden von den Anwendungen, die den Identitäts-Manager nutzen, selbst verwaltet. Eingehende Nachrichten werden mittels des Observer-Prinzips an die, den Manager

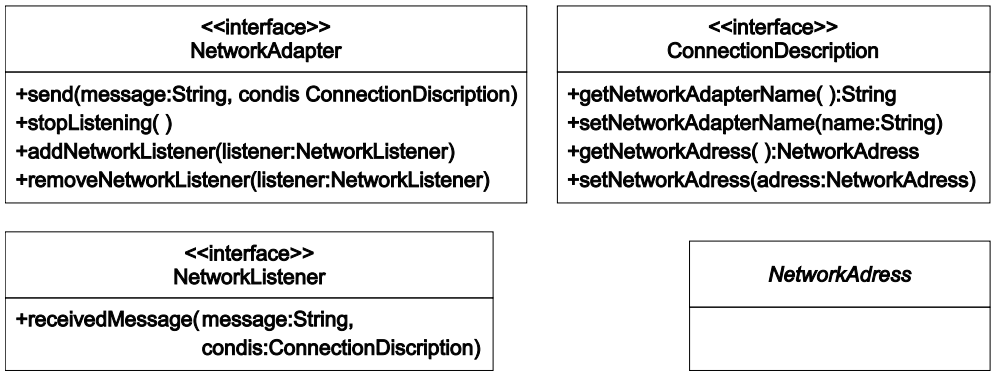

Abbildung 4.9: Klassen des Netzwerk-Paketes der onefC-Architektur

nutzenden Anwendungen übermittelt. Ist eine zeitliche Synchronisation für die Applikation notwendig, muss diese selbst vorgenommen werden. Dies ist zum Beispiel beim Einsatz des Identitäts-Managers bei der Erzeugung von personalisierten Webseiten notwendig. Da die asynchrone Kommunikation der allgemeinere Fall ist, also die synchrone einschließt, grenzt diese Form der Abstimmung der Nachrichten die Einsatzmöglichkeiten nicht ein.

Ein weiterer Vorteil dieser Art des Nachrichtenaustausches ist, dass die Anfrage und ihre Antwort nicht mit dem gleichen Transportprotokoll übertragen werden müssen. Es ist also jederzeit möglich dieses zu wechseln, ohne dass die darüber liegende Anwendung angepasst werden muss.

4.4.5 Persistenz

Die Aufgabe der Persistenz-Schicht ist es die vom onefC-Manager verwalteten Identitäten dauerhaft zu speichern. Bei der Art der Speicherung ist dem sensiblen Charakter der Daten Rechnung zu tragen. Dies bedeutet, dass die Daten vor dem Zugriff nicht autorisierter Personen zu schützen sind. Dies kann durch Verschlüsselung der abgelegten Identitäten erreicht werden. Ein zweiter Punkt, den die Persistenz-Schicht lösen muss ist die der Portabilität. Da die digitalen Identitäten im Allgemeinen an eine Person gebunden sind und nicht an einen Computer, sollten die Daten so gespeichert werden, dass sie diese Daten mit sich führen kann. Dies kann zum Beispiel durch den Einsatz von Chip-Karten als Speichermedium erreicht werden. Es ist aber auch der Einsatz von Persönlichen Digitalen Assistenten (PDA) denkbar. Diese könnten durch die Verwendung von Middleware-Systemen in die onefC-Architektur eingebunden werden (vergl. [Köh00]).

4.4.6 Übersetzungsdienst

Um den Identitäts-Manager noch flexibler zu machen, ist der Einsatz von Übersetzungsdiensten vorgesehen. Ein solcher Dienst, soll Anwendungen unterstützen,

Daten von anderen Applikationen zu nutzen. Außerdem können die Übersetzungsdienste auch die Wiederverwendbarkeit der bereits im Profil enthaltenen Attribute erhöhen. Dies ist möglich, da die Inhalte der Profil-Attribute in der Extensible Modeling Language kodiert sind. Ein Übersetzungsdienst hat deshalb lediglich die Aufgabe eine XML-Repräsentation in eine andere zu überführen. Dazu kann zum Beispiel die Sprache XSLT verwendet werden. In ihr werden sogenannte Stylesheets verfasst, mit denen XML-Daten eines Schemas in Daten eines anderen transformiert werden können. Die einzelnen Übersetzungsdienste werden durch eine einheitliche Schnittstelle (`TranslatorService`) angesprochen und dynamisch in die Architektur eingebunden. Die verwendete Schnittstelle zum Übersetzungsdienst (vergl. Abbildung 4.10) definiert drei Methoden. Es gibt jeweils eine Methode, um die Schemata zu ermitteln, zwischen denen die Transformation stattfindet. Die Methode „`translate`“ führt dann die eigentliche Übersetzung des Inhaltes eines Attributes durch.

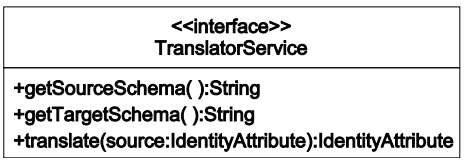

Abbildung 4.10: Schnittstelle des Übersetzungsdienstes der onefC-Architektur

4.5 Protokoll-Ansatz

Um den Identitäts-Manager im Netzwerk testen zu können, ist eine Spezifikation eines Protokolls zum Austausch von Identitätsdaten notwendig. Die hier vorgestellten Nachrichten sollen kein vollwertiges Identitäts-Kommunikations-Protokoll aufbauen. Sie sind vielmehr zum Zweck der Evaluation der Management- und Identitäts-Infrastruktur gedacht. Die folgenden Nachrichten definieren ein einfaches Frage-Antwort-Protokoll. Auf jeden der Anfragetypen antwortet der Identitätsmanager mit dem Aussenden einer passenden Identität. Lediglich die Aufforderung zum Speichern eines Attributes (Storage-Request) bleibt unbeantwortet.

Alle Nachrichten, die vom Identitäts-Manager verarbeitet werden können, sind von der Klasse `MessageObject` abzuleiten. Für eine exakte Diversifizierung sind die Nachrichtenobjekte noch in Anfragen (`RequestObject`) und Antworten (`AnswerObject`) unterteilt (siehe Abbildung 4.11).

4.5.1 Identity- und Foreign-Identity-Request

Die `IdentityRequest`-Typen stellen die Anfrage nach einer vollständigen Teilidentität in einem Kontext dar. Mit diesen Nachrichten, können lokale und fremde Identitäten angefragt werden. Die Bezeichnung „lokal“ steht dabei für Identitäten, die

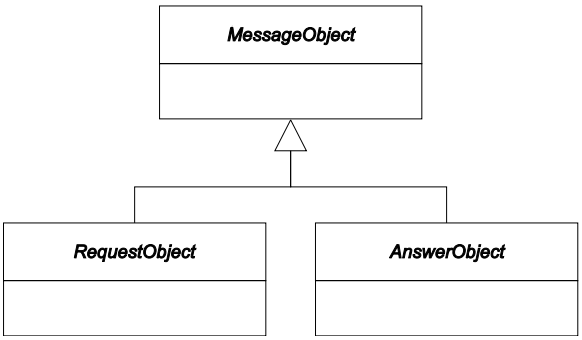

Abbildung 4.11: Hierarchie der abstakten Nachrichten-Klassen des onefC-Managers

durch den Identitäts-Manager selbst erzeugt und verwaltet werden. Fremde Identitäten hingegen sind diejenigen, die der Identitäts-Manager durch eigene Anfragen von anderen Nutzern erhalten hat. Die Weitergabe eigener Daten kann bei der Übertragung der eigenen Attribute eingeschränkt werden, um die Kontrolle des Nutzers durch den `ForeignIdentityRequest` nicht außer Kraft zu setzen.

Wird bei einer lokalen Anfrage kein Adressat angegeben, so entscheidet der Sicherheitsdienst des Identitäts-Managers, welcher Identitätsbaum zur Beantwortung herangezogen wird.

4.5.2 Attribute-Request

Um gezielt in einem Kontext Attribute einer Identität abzufragen, gibt es den `AttributeRequest`. Dieser enthält eine Liste mit Attributen, die als Maske dienen. In die Antwort werden nur die Attribute eingetragen, die die gleichen Schemata und Namen wie die Masken-Attribute besitzen. Dabei gibt es die Möglichkeit einen Platzhalter zu verwenden, der für beliebige Einträge steht.

4.5.3 Storage-Request

Der `StorageRequest` repräsentiert die Bitte eines fremden Identitäts-Managers, ein Attribut in der lokalen Identität abzuspeichern. Dieses Attribut wird in der Anfrage mitübertragen. Sollte die Anfrage nach Speicherung akzeptiert werden, so wird der Absender der Nachricht als Urheber im Fremdattribut vermerkt.

4.5.4 Identity-Answer

Mit der `IdentityAnswer` antwortet ein Identitäts-Manager auf eine gestellte Anfrage. Sie enthält die Teilidentität für den aktuellen Kontext. Die in ihr enthaltenen Attribute richten sich zum einen nach dem Anfragetyp und zum anderen danach, welche Einträge vom Sicherheitsdienst autorisiert wurden.

Kapitel 5

Implementierung der Infrastruktur Komponenten

An dieser Stelle soll die Implementierung der in Kapitel 4 entwickelten Konzepte vorgestellt werden. Zunächst werden die Umsetzungen des Identitäts-Baumes als Objekt- und XML-Struktur vorgestellt. Danach wird auf die Toolkits eingegangen, mit deren Hilfe nicht triviale Funktionen auf den Identitäten ausgeführt werden können. Im Anschluss wird dann die Implementierung der Management-Komponenten dargestellt.

5.1 Der Identitäts-Baum

Die onefC-Identität wird von den Komponenten des Managers in zwei verschiedenen Repräsentationen verwendet. Zum einen gibt es die Umsetzung der Identität in eine Objektstruktur (vergl. Abbildung 4.4). Dies ist die Repräsentation, die der Anwendungsebene zur Verfügung steht. Die andere Umsetzung ist die Abbildung auf ein XML-Dokument. Diese Dokumente werden zum Datenaustausch verwendet, da sie unabhängig von der verwendeten Programmiersprache sind.

Obwohl beide Formen der Darstellung die gleichen Inhalte repräsentieren können, gibt es einen wesentlichen Unterschied. Die XML-Abbildung gibt lediglich an, wie die Identität aufgebaut ist und welche Inhalte sie hat. Die Repräsentation als Objektstruktur definiert als Datentyp zusätzlich noch Methoden zum Datenzugriff, zur Manipulation der Daten und zu deren Inspizierung. Diese Methoden können in den einzelnen Implementierungen variieren, ohne die inhaltlichen und strukturellen Informationen der Daten zu beeinflussen.

5.1.1 Programmiersprachliche Umsetzungen

Erstellung der Objektstruktur

Der onefC-Manager stellt über seine Schnittstelle Identitäten als `Identity`-Objekte zur Verfügung. Diese Klasse implementiert den abstrakten Baum über dessen Kno-

ten. Jeder dieser einzelnen Knoten besitzt drei Speicher. Im ersten (`KontextList`) sind die Kontexte, für die der Knoten gültig ist, enthalten. Der zweite (`Attribute-List`) beinhaltet die Attribute, die dieser Teil-Identität hinzugefügt wurden. Über den dritten Speicher (`IdentityList`) sind die Kinderknoten referenziert. Außerdem sind die Merkmale der Identität, wie zum Beispiel der Aliasname oder der eindeutige Bezeichner, in entsprechenden Variablen gespeichert. Um vererbte Attribute leichter bestimmen zu können, ist zusätzlich noch eine Referenz auf den Elternknoten gegeben. Die vom Identitätsknoten bereitgestellten Methoden beziehen sich immer nur auf die aktuelle Teil-Identität. Methoden, die mehrere Knoten bearbeiten oder sich auf die enthaltenen Attribute beziehen, sind in entsprechende Toolkits ausgelagert. Damit wird die Realisierung des Baumes über seine Teile konsequent eingehalten. Die vom `Identity`-Objekt bereitgestellte Funktionalität dient vor allem zum Erhalt, zum Hinzufügen und zur Manipulation der enthaltenen Komponenten.

In die Liste der Kinder der Identität kann nicht direkt eingefügt werden. Dies ist nur über die Methode `createSubIdentity()` möglich. Durch diese Art der Implementierung baut sich der Baum nur aus sich selbst heraus auf. Damit kann gewährleistet werden, dass Attribute, die über alle Teil-Identitäten des Baumes gleich bleiben sollen, diese Anforderung auch erfüllen. So erzeugt die Methode eine neue Teil-Identität, mit dem gleichen Identifier wie beim Elternknoten. Setzt die Referenz zum Elternknoten und fügt erst dann das neue `Identity`-Objekt in die Liste der Kinder ein.

Diese Art des Aufbaus des Identitäts-Baumes wird jedoch bei der Erstellung der Objektstruktur aus der XML-Repräsentation umgangen. Die hier verwendete und von `Identity` abgeleitete `ParsedIdentity` erlaub durch entsprechende Methode das direkte Einfügen der Kinder. Dies ist an dieser Stelle unproblematisch, da durch das XML-Dokument die Struktur und die Inhalte bereits vorgegeben sind. Nach außen wird die `ParsedIdentity` jedoch nur als ihre Elternklasse weitergereicht, so dass im weiteren Umgang der Schutz vor direktem Einfügen wieder gewährleistet ist.

Komponenten innerhalb der Identität

Bezeichner Die Bezeichner, die in den Komponenten der Identität verwendet werden, sind alle von der `Identifier`-Klasse abgeleitet. Diese definiert die zwei abstrakten Methoden `equals()`[1] und `getUniqueObject()`, die allen Bezeichnern gemein sind. Dies wird gemacht, damit die Komponenten auch ohne Kenntnis der konkreten Klasse mit den verschiedenen Bezeichnern umgehen können. Zur Zeit gibt es zwei Implementierungen von Bezeichnern. Zum einen die `UniqueIdentifier` und zum anderen die `StringIdentifier`. Die `UniqueIdentifier` haben als eindeutiges Objekt eine UUID (siehe 2.1.1), die beim Instanziieren des Objektes neu erzeugt wird. Die Implementierung der UUID ist aus dem Projekt JAXTA entnommen. Sie wurde dahingehend verändert, dass die verwendeten Zufallszahlen den Anforderungen für sicherheitsrelevante Anwendungen genügen (vergl. [ECS94]). Dies wurde durch den

[1] Die Methode `equals()` der `Identifier`-Klasse unterscheidet sich in soweit von der entsprechenden Methode der `Object`-Klasse, als dass sie nur `Identifier` als Parameter akzeptiert.

Einsatz der `SecureRandom`-Klasse erreicht. Der `StringIdentifier` wird eingesetzt, wenn zum Beispiel URLs als eindeutige Bezeichnung verwendet werden sollen. Bei dessen Einsatz hat jedoch der Benutzer auf die Eindeutigkeit zu achten (Vertragsmodell).

Kontext Die `Context`-Klasse ist durch das enthaltene `Identifier`-Objekt bestimmt. Dieses kann nur einmal beim Erzeugen des Kontextes gesetzt werden. Danach besteht keine Möglichkeit es zu ändern. Zusätzlich kann noch ein Aliasname vergeben werden, der in einer entsprechenden Variable gespeichert wird. Dieser kann durch den Anwender verändert werden, da er den Kontext nicht definiert. Der Alias ist vielmehr eine Anmerkung zum besseren Verständnis.

Attribute Alle Attribute, die in das Profil einer Teil-Identität eingefügt werden sollen, müssen von der abstrakten Klasse `IdentityAttribute` erben. Dies ist notwendig, da die Methode `addAttribute()` des Identitäts-Knotens nur Objekte diesen Typs als Parameter akzeptiert. Außerdem beinhaltet diese Basisklasse die allen Attributen gemeinen Parameter und Variablen. Dies ist zum Beispiel ein eindeutiger Bezeichner. Diese sind zur Zeit drei Konkretisierungen eines Attributes. Es gibt das `ProfileAttribute`, das `ForeignProfileAttribute` und das `SocialIdentity-Attribute`. Das Attribut einer sozialen Identität definiert diese durch den verwendeten eindeutigen Bezeichner. Die beiden Profil-Attribute beinhalten Variablen für Schema, Name und Inhalt der Profildaten. Als Name und Inhalt können beliebige Objekte verwendet werden. Diese sollten jedoch XML Elemente bzw. deren Namen repräsentieren. Sie müssen durch die Methode `toString()` als XML-Fragment bzw. Zeichenkette formatiert werden können, um sie unabhängig von der gewählten Implementierung versenden oder abspeichern zu können. Das `ForeignProfileAttribute` speichert zusätzlich noch den Bezeichner der Identität, die das Attribut übermittelt hat.

Verbreitung von Informationen über Aktualisierungen

Da Identitäten gleichzeitig von mehreren Anwendungsteilen benutzt werden können, muss es ein Konzept zur Verbreitung von Informationen über Veränderungen der Inhalte geben. Die onefC-Identität realisiert dieses durch die kaskadierte Anwendung des Beobachter-Musters (vergl. Abschnitt 2.3.4). Jedes Element und die Identität selbst bieten die Möglichkeit, dass sich Objekte, die die `IdentityElementListener`-Schnittstelle implementieren, als Beobachter registrieren. Diese werden dann jeweils über Änderungen des Elements informiert. Für die Identität gilt, dass sie sich nicht nur dann ändert, wenn ihr Elemente hinzugefügt oder diese wieder aus ihr entfernt werden, sondern auch dann, wenn sich eines ihrer Elemente ändert. Deshalb registriert sich die Identität bei jedem ihrer Elemente als Beobachter und reicht eine sie erreichende Änderungsinformation an ihre Beobachter weiter.

5.1.2 Umsetzung in XML

Um die Abbildung der programmiersprachlichen Umsetzung der Identität auf die serielle XML-Repräsentation zu erleichtern, sind alle Komponenten von der abstrakten Klasse `BasicIdentityElement` abgeleitet. Diese definiert die Methode `toXML()`, die nach jeweiliger Implementierung eine Komponente in ihre XML-Repräsentation überführt. Durch Verschachtelung dieser Aufrufe können auch komplexere Objekte leicht als XML-Dokument dargestellt werden (vergl. Abbildung 5.1). Jede Klasse muss nur die durch sie gemachten Erweiterungen hinzufügen und die Ausgabemethode der internen Objekte aufrufen. Dabei muss jede Komponente die Methode `toXML()` so implementieren, dass die Ausgabe entsprechend dem für Identitäten festgelegten Schema entspricht (siehe Anhang A.1). Auch Änderungen am Schema sind so leicht durchzuführen, da lediglich die einzelne Komponente an das neue Schema angepasst werden muss.

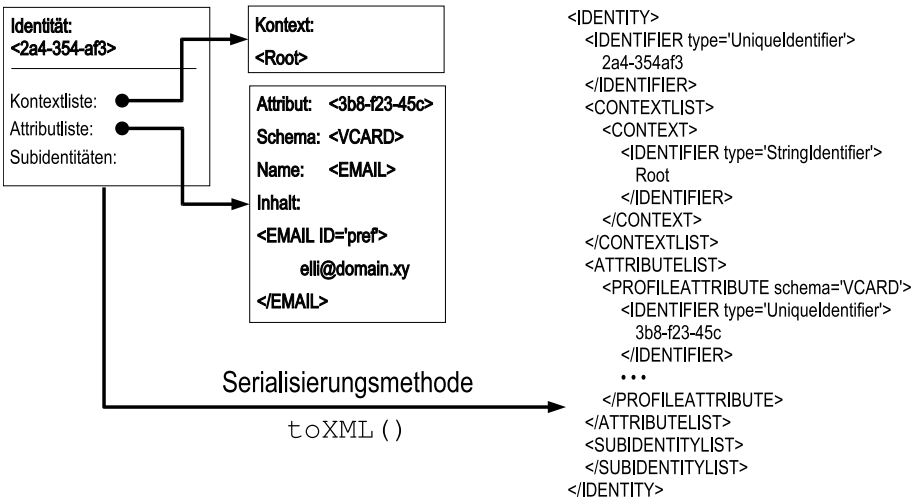

Abbildung 5.1: Serialisierung der Objekt- in die XML-Struktur

5.2 Einsatz von Toolkits

Die einzelnen Teil-Identitäten bieten jeweils die Funktionalität, die den aktuellen Knoten betrifft. Komplexere Methoden, die zum Beispiel den gesamten Identitäts-Baum bearbeiten, können aus diesen Methoden zusammengesetzt werden. Um die Funktionalität nicht immer neu implementieren zu müssen, sind diese in einem `IdentityToolkit` zusammengefasst. Zum erweiterten Umgang mit Attributen und Attributlisten existiert ein entsprechendes `AttributeToolkit`.

5.2.1 Das Identitäts-Toolkit

Die im `IdentityToolkit` enthaltenen Methoden, beziehen sich auf einen ganzen Identitäts-Baum oder Teile von ihm. Es wird speziell für einen Knoten instanziiert und bietet folgende Methoden:

- `getContextListOfIdentityTree()`
  zum Erlangen der Liste aller Kontexte des Teilbaumes;

- `getExtendedAttributes()`
  zum Erhalt der Liste der geerbten Attribute;

- `getIdentityOfIdentityTree()`
  zum Erlangen des zu einem Kontext gehörenden Identitäts-Knotens aus einem Teil-Baum;

- `getProfileAttribute()`
  zum gezielten Auslesen eines Profilattributes anhand des Schemas und des Namens;

5.2.2 Das Attribute-Toolkit

Das `AttributToolkit` stellt statische Methoden zum Umgang mit Attributlisten bereit. Die Methoden werden vor allem zum Aufbau der Identitäts-Antworten verwendet. Die Methoden sind im einzelnen:

- `mergeWithPriority()`
  zum Vereinigen zweier Attributlisten, wobei Attribute mit gleichem Bezeichner der ersten Liste Vorrang haben (Umsetzung des Sichtbarkeitskonzepts);

- `attributeMatchPattern()`
  zum Filtern einer Attributliste anhand einer Liste von Mustern;

- `removeOutOfTime()`
  zum Filtern von Attributen, deren Lebensdauer abgelaufen ist oder zu einem angegebenen Zeitpunkt abläuft;

5.3 Die Identitäts-Management Komponenten

An dieser Stelle wird die Implementierung der in Abschnitt 4.4 vorgestellten Komponenten des onefC-Managers beschrieben. Dabei ist sie in die zentrale Management-Komponente, die von ihr verwendeten Dienstkomponenten und die Umsetzung des Kommunikationsprotokolls unterteilt.

5.3.1 onefC-Identitäts-Manager

Die zentrale Management-Komponente beinhaltet zwei Speicher, je einen für die eigenen und die fremden Identitäten. Zudem werden die beim Manager registrierten `IdentityManagerListener` und `MonitoringServices` in Vektoren abgelegt.

Bei der Instanziierung werden der `SecurityService`, der `PersistenceService`, der `ProtocolfileMonitor` und die `NetworkAdapter` erzeugt. Dazu werden die in den Projektvorgaben gemachten Einstellungen gelesen und entsprechende dynamische Fabriken instanziiert (vergl. Abschnitt 5.3.2). Diese erzeugen dann die konkreten Objekte. Mit der Methode `getPersistentIdentities()` müssen gegebenenfalls die gespeicherten lokalen Identitäten in den Manager geladen werden. Damit ist die Initialisierung der Management-Komponente abgeschlossen.

Die Methoden zur Registrierung und Deregistrierung werden an die entsprechenden Vektoren weitergeleitet. Die verfügbaren Adapternamen, werden von dem entsprechenden Fabrikobjekt geliefert und müssen ebenfalls nur weitergereicht werden. Zur Herausgabe der Liste der eigenen bzw. fremden Identitäten, werden die Einträge der Vektoren in neue `IdentityList`-Objekte eingefügt. Zum Laden der persistenten Identitäten, wird vom `PersistenceService` die Liste der gespeicherten Instanzen angefordert und in einer Iteration dem Speichervektor hinzugefügt. Analog werden die vom Manager verwalteten Identitäten einzeln dem Persistenzdienst zum Speichern übergeben. Mit der Methode `createRootIdentity()` wird eine neue lokale Identität erzeugt und dem Speichervektor hinzugefügt. Dabei erhält sie einen Kontext mit der Bezeichnung „*Root*“, damit die erzeugte Instanz eine vollwertige Identität darstellt. Soll eine Identität gelöscht werden, so ist nicht nur der Eintrag im lokalen Speicher zu entfernen, sondern auch eine eventuell vorhandene gespeicherte Fassung.

Eingehende Nachrichten werden vom Identitäts-Manager über sogenannte Handler-Methoden bearbeitet (siehe Abbildung 5.2). Dies hat den Vorteil, dass neue Nachrichten-Typen leicht eingebunden werden können. Dazu müssen dem Manager lediglich entsprechende Methoden hinzugefügt werden. Die einzelnen Handle-Methoden nutzen bei ihrer Abarbeitung weitere allgemeingültige Methoden. Dazu zählen die Methode zum Überprüfen der Weitergabeberechtigung (`checkReadPermissionOfListAttributes()`), oder Toolkit-Methoden, wie zum Beispiel `removeOutOfTime()` zum Entfernen von abgelaufenen Attributen. Diese Ausgliederung wiederkehrender Funktionalität soll ebenfalls die Erweiterbarkeit erleichtern.

Zusätzlich wird in den Handler-Methoden auch die Pseudonymisierung vorgenommen. Dazu greifen sie auf ein `PseudonymResolver`-Objekt zu, welches die Auflösung von Pseudonym in original Identität und umgekehrt vornimmt. Mit dessen Hilfe wird vor der Bearbeitung einer eingehenden Nachricht überprüft, ob die in der Nachricht enthaltene Identität pseudonymisiert wurde und führt gegebenenfalls eine Abbildung auf die original Identität durch. Vor dem Versand einer Antwort wird, wenn dies im Resolver eingetragen ist, die inverse Transformation vom Original zum Pseudonym durchgeführt.

Die Methode `askRemoteIdentity()`, die Anfragen an andere Identitäts-Manager

stellt, kann sehr allgemein gehalten werden. Dies ist dadurch möglich, dass die ihr übergebenen RequestObjekte selbst den Nachrichteninhalt definieren und formatieren. Die Management-Komponente muss diesen lediglich durch Aufruf einer vordefinierten Methode (`getMessageBody()`) in die Identitäts-Manager-Nachricht einfügen.

Den onefC-Identitäts-Manager gibt es in zwei Varianten. Er ist einmal als normales Objekt verfügbar und einmal als Singleton. Je nach Verwendungszweck kann die eine oder andere Variante genutzt werden. So kann man sich zum Beispiel vorstellen, dass ein Webserver das Singleton verwendet, um von allen Servlets oder Java-Server-Pages aus auf einen gemeinsamen Manager zuzugreifen. Nutzt eine Anwendung den onefC-Manager lediglich zur Abfrage anderer Manager, ist auch der Einsatz einer separaten Instanz möglich.

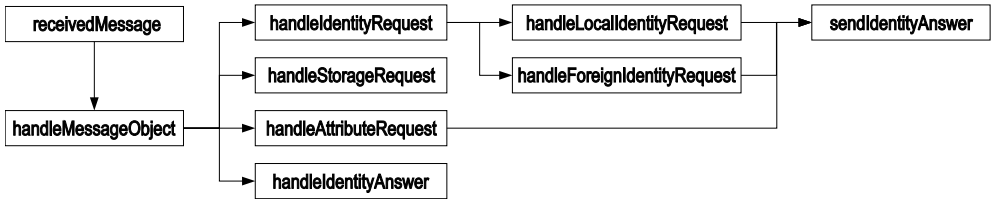

Abbildung 5.2: Nachrichtenverarbeitung der Manager-Komponente

5.3.2 Dienst-Komponenten

Monitordienste

Die Umsetzung der onefC-Architektur bietet zwei Implementierungen des Monitordienstes an. Zum einen den `ProtocolfileMonitor` und zum anderen den `PanelMonitor`. Von der Management-Komponente wird jedoch lediglich der Monitor zum Erzeugen einer Protokolldatei verwendet. Der Panel-Monitor hingegen stellt eine einfache Möglichkeit der Einbindung des Protokolls in eine grafische Oberfläche dar.

Protokolldatei-Monitor Der `ProtocolfileMonitor` ist ein Monitor, der sein Protokoll in einer Datei speichert. Um Zugriffskonflikte zu vermeiden, ist der er nach dem Singleton-Muster konzipiert (vergl. Abschnitt 2.3.2). Damit ist immer nur eine Instanz des Monitors in der virtuellen Maschine vorhanden und konkurrierende Zugriffe können nicht auftreten. Ob beim Start des Protokolldatei-Monitors das letzte Protokoll überschrieben wird oder die aktuellen Daten an die Datei angehängt werden, kann der Benutzer selbst entscheiden. Diese Einstellung kann er in den Projektvorgaben machen (vergl. Abschnitt 5.3.2).

Beim Instanziieren des `ProtocolfileMonitors` werden zunächst die Angaben zum Dateinamen und dem Fortschreibemodus aus den Projektvorgaben gelesen. Sollten die Einträge nicht vorhanden sein, so werden standardisierte Werte verwendet. Danach wird die Protokolldatei geöffnet. Jeder Aufruf der `notice()`-Methode schreibt

die übergebene Nachricht zusammen mit einem Zeitstempel direkt in die Datei. Das Protokoll wird erst dann geschlossen, wenn das Monitorobjekt vom System gelöscht wird. Dazu ist die von `Object` geerbte Methode `finalize()` überschrieben worden. Damit ist sichergestellt, dass das Protokoll fortgeschrieben werden kann, solange eine Referenz auf den Monitor existiert.

Panel-Monitor Um die Integration des Monitoringdienstes in grafische Oberflächen zu erleichtern, gibt es in der onefC-Implementierung den `PanelMonitor`. Dieser Monitor ist eine Erweiterung des in JAVA vorhandenen `JPanel` und stellt somit eine SWING-Komponente dar. Er enthält ein mit „*Identity Manager Monitor*“ überschriebenes Textfeld (siehe Abbildung 5.3). Die über `notice()` übermittelten Nachrichten werden mit einem aktuellem Zeitstempel versehen und in das Textfeld geschrieben. Als vorgefertigte grafische Komponente, lässt er sich leicht in beliebige Anwendungen integrieren.

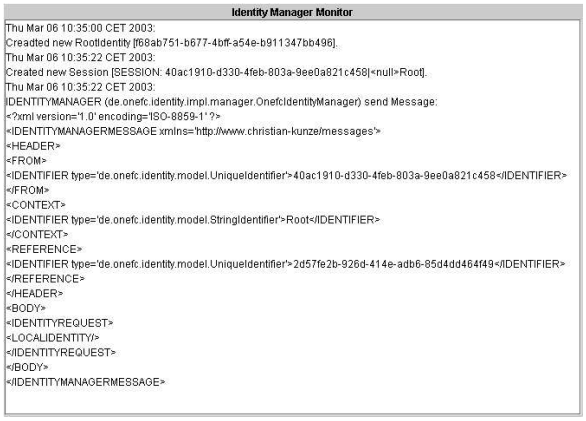

Abbildung 5.3: Grafische Oberfläche des Panel-Monitors

Network-Adapter

Der onefC-Identitäts-Managers kann über Adapter auf beliebige Netzwerke zugreifen. In der aktuellen Implementierung ist ein Adapter zur Kommunikation über TCP/IP-Netzwerke enthalten. Dieser `SocketNetworkAdapter` kapselt und verwaltet zwei Teilkomponenten zum Senden und Empfangen von Nachrichten. Er ist eine Erweiterung des `AbstractNetworkAdapter`, der die Registrierung und das Entfernen der `NetworkListener` implementiert.

Empfangen von Nachrichten Um Nachrichten über das Netz empfangen zu können, erzeugt der `SocketNetworkAdapter` bei seiner Instanziierung einen `Thread` mit einem `ReceivingObject`. Dieses dient als Server-Komponente und wartet auf Nachrichten von anderen Managern. Durch Übergabe einer Referenz auf die beim `SocketNetworkAdapter` gespeicherte Liste der `NetworkListener`, stehen diese auch dem

Server-Objekt zur Verfügung. Dieser kann dadurch die eingehenden Nachrichten selbst an die registrierten Objekte weiterleiten. Dies geschieht, indem die Server-Komponente für jeden `NetworkListener` der Liste einen eigenen `Thread` mit einem `InformObject` erzeugt. Dieser Umweg ist aus Performanzgründen notwendig. Würden die entsprechenden Methoden der Listener nacheinander im `ReceivingObject` aufgerufen, so wäre der Server während der Abarbeitung blockiert. Die in dieser Zeit ankommende Nachrichten könnten also nicht verarbeitet werden.

Der vom `ReceivingObject` verwendete Port kann individuell durch die Methode `setServerPort()` angegeben werden. Als Standardwert ist der Port 2108 eingestellt. Zum Beenden des Servers wird die Methode `stopListening()` verwendet.

Senden von Nachrichten Für jede zu sendende Nachricht, erzeugt der `Socket-NetworkAdapter` einen `Thread` mit einem `SendingObject`. Beim Erzeugen werden Name oder IP-Adresse, Port und Nachricht an die Sendekomponente übergeben. Diese baut mit diesen Daten eine Verbindung zum Kommunikationspartner auf und überträgt die Nachricht. Danach wird die Verbindung und der `Thread` beendet.

Persistenzdienst

Der Persistenzdienst, den die onefC-Implementierung zur Zeit verwendet, ist der `FilePersistenceService`. Dieser speichert die Identitäten des Nutzers als XML-Dokument im Dateisystem. Dazu werden die an ihn übergebenen Identitäts-Objekte durch Aufruf der `toXML()`-Methode in die XML-Repräsentation überführt. Danach werden sie unter dem achtstelligen Hashwert ihres Bezeichners und der Endung „*.identity*“ als Textdatei gespeichert. Die noch unverschlüsselte Speicherung der Daten ist zu Verifizierungs- und Testzwecken gewählt worden. Sie sollte zu einem späteren Zeitpunkt durch eine den Sicherheitsbedürfnissen des Nutzers Rechnung tragende Implementierung des `PersistenceService` ersetzt werden.

Sicherheitsdienst

Der Sicherheitsdienst (`SecurityService`) ist lediglich als Attrappe oder Dummy implementiert, da der Sicherheitsdienst Teil einer anderen Arbeit sein wird.

Dies Implementierungsform bedeutet, dass der `DefaultSecurityService` unabhängig von den eingegebenen Parametern immer fest eingestellte Ergebnisse liefert. Die Autorisation und das Prüfen eines Logins werden immer positiv entschieden. Die Auswahl eines Identitätsbaumes ist immer der erste Eintrag der übergebenen Liste.

Hilfskomponenten

Dynamische Fabriken Die `DynamicFactory` ist eine Klasse, die dem Design-Muster der Fabrik folgt (vergl. Abschnitt 2.3.1). Dieses sagt aus, dass ein Fabrik-Objekt beliebige Objekte eines speziellen Typs erzeugen kann. Die dynamische Fabrik

bekommt jedoch erst bei ihrer Instanziierung mitgeteilt, welche Objekt-Typen sie erzeugen soll. Sie sucht dazu in einem angegebenen Verzeichnis nach Klassen, die eine gemeinsame Oberklasse haben oder ein spezielles Interface implementieren. Damit ist sie flexibel einsetzbar.

Die in der onefC-Architektur eingesetzte `DynamicFactory` kann sowohl normale Objekte erzeugen, als auch Singletons. Dazu überprüft sie das Vorhandensein der Methode `instance()` bei Klassen, die nicht auf dem gewöhnlichen Weg erzeugt werden können und ruft diese gegebenenfalls auf. Sie wird zum Beispiel zum Instanziieren der Netzwerk-Adapter verwendet. Damit ist das Einbinden neuer Netzwerk-Adapter in die onefC-Architektur wesentlich vereinfacht.

Projekt Einstellungen Die Einstellungen und Vorgaben des onefC-Identity-Managers werden über eine Properties-Datei angegeben. Die enthaltenen Einträge stehen den Komponenten des Managers über die `ProjectProperties`-Klasse zur Verfügung. Diese Komponente ist nach dem Singleton-Muster konzipiert (vergl. Abschnitt 2.3.2). Dies bedeutet, dass es von der Klasse immer nur eine Instanz pro virtueller Maschine gibt, die von allen anderen Komponenten benutzt wird. Der Ort, wo sich die Datei mit den Einstellungen befindet, wird dem Manager über eine zusätzliche Systemeigenschaft mitgeteilt. Dazu wird beim Aufruf eines Programms, das den Manager benutzt, durch Angabe von `-Dproperties.root="`*`<Verzeichnispfad>`*`"` der zusätzliche Systemparameter übergeben. Die Datei muss den Namen `onefc.properties` tragen und enthält die jeweiligen Einträge als Name/Wert-Paar (siehe Tabelle 5.1). Sollten Einträge fehlen, so werden Standardwerte verwendet.

onefC-XML-Syntaxanalysierer Der in der onefC-Architektur verwendete Syntaxanalysierer (`OnefcParser`) baut auf einem durch einen allgemeinen XML-Parser erzeugten DOM-Baum auf. Er kann sowohl über das Netzwerk erhaltenen Nachrichten als auch in XML repräsentierte Identitäten analysieren. Dazu stellt er die beiden Methoden `parseMessage()` und `parseIdentity()` bereit, die entsprechende Identitäts- bzw. Nachrichtenobjekte erzeugen.

Der interne Aufbau des Analysierers ist so gestaltet, dass für jede Komponente, die in einer Nachricht bzw. XML repräsentierten Identität vorkommen kann, eine eigene Methoden zur Extraktion aus dem DOM-Baum existiert. Dies hat den Vorteil, dass sie zum einen für alle Elemente verwendet werden können und zum anderen, dass komplexe Elemente durch einfaches Verschachteln analysiert werden können. So sieht zum Beispiel das Extrahieren eines Kontextes in stilisierter Form folgendermaßen aus:

```
extractContext():Context
{
  time-to-live = parseLong(extractString():String):Long;
  alias = extractString():String;

  identifier = extractIdentifier():Identifier;
```

Name	Wert
identity.file.path	Pfad zum Verzeichnis der Identitäten
protocol.append	Einstellung zur Fortschreibung des Protokolls
protocol.filename	Name der Protokolldatei
service.persistance.name	Klassenname des Persistenzdienstes
service.persistance.package	Paket des Persistenzdienstes
service.persistance.root	Stammverzeichnis des Persistenzdienstes
service.security.name	Klassenname des Sicherheitsdienstes
service.security.package	Paket des Sicherheitsdienstes
service.security.root	Stammverzeichnis des Sicherheitsdienstes
service.network.package	Paket der Networkadapter
service.network.root	Stammverzeichnis der Networkadapter

Tabelle 5.1: Mögliche Einträge der Properties-Datei

```
  {
    type = extractString():String;
    value = extractString():String;
  }
}
```

Diese Art der Verschachtelung der Extraktionsmethoden erlaubt es, einen kompletten Identitäts-Baum durch einfache Rekursion zu traversieren und somit zu analysieren. Ein weiterer Vorteil ist, dass der `OnefcParser` leicht erweitert werden kann, um neue Nachrichten oder Identitäts-Elemente hinzuzufügen.

5.3.3 Ansatz des Kommunikationsprotokolls

Wie in Abschnitt 4.5 beschrieben, sind alle Protokoll Nachrichten von den Klassen `RequestObject` und `AnswerObject` abgeleitet. Damit können alle Anfragen durch die Management-Komponente verarbeitet und verschickt werden. Dazu implementieren sie die abstrakte Methode `getMessageBody()`, die die Anfrage in XML überführt. Wie die Syntax der unterschiedlichen Anfrage beschaffen sein muss, ist in einem Schema definiert (siehe Anhang A.2). Zusätzlich definiert jeder Nachrichtentyp eigene Methoden, die den Umgang mit ihm erlauben.

Die Anfragen sind dabei so gestaltet, dass auf sie nach ihrer Erzeugung lediglich lesend zugegriffen werden kann. Damit wird verhindert, dass die enthaltenen Elemente verändert werden und so Seiteneffekte auftreten könnten. Die Anfragen nach lokalen

und fremden Identitäten sind in einer Klasse zusammengefasst. Die Unterscheidung wird durch ein spezielles Attribut beim Instanziieren des Objektes gemacht.

Das `IdentityAnswerObject` ist die einzige im Protokollansatz definierte Antwort auf eine Anfrage. Sie ist von `AnswerObject` abgeleitet und enthält die Identität, die durch die korrespondierende Anfrage angefordert wurde.

Alle Nachrichten besitzen eine eindeutige Referenz, die in dieser Umsetzung durch einen `UniqueIdentifier` repräsentiert wird. Damit können Anfragen und Antworten einander zugeordnet werden. Dies ist notwendig, da das Protokoll zeitlich nicht synchronisiert ist.

Kapitel 6

Anwendungsszenarien

Dieses Kapitel stellt ausgewählte Szenarien vor, in dem Identitäts-Management eingesetzt werden kann. Neben der Beschreibung des Anwendungsfalls wird jeweils eine mögliche Lösung mit dem onefC-Identitäts-Manager skizziert.

6.1 Authentifizierung

Szenario-Beschreibung

Authentifizierung bezeichnet allgemein einen Prozess, mit dem ein Grad an Vertrauen in eine Aussage oder Behauptung etabliert wird. In Netzwerken wird häufig die Authentifizierung von Identitäten vorgenommen. Hierbei möchte eine Partei mit relativer Sicherheit erkennen, ob ein Kommunikationspartner ihm schon bekannt ist. [Cla99]

In diesem Szenario soll ein Server die Nutzer, die sich bei ihm zuvor registriert haben, automatisch authentifizieren. Das vom Betreiber verlange Sicherheitsniveau soll gleich dem eines passwortbasierten Verfahrens sein.

Lösung mit dem onefC-Identitäts-Manager

Eine Lösung mit der onefC-Identitäts-Infrastruktur könnte bei Verwendung eines tokenbasierten Verfahrens folgendermaßen aussehen (siehe Abbildung 6.1):

Registrierung Der erste Teil des Verfahren ist die Registrierung des Nutzers beim Server. Hierbei werden zunächst die erforderlichen Daten zwischen Server und Anwender ausgetauscht. Dieser Schritt kann, muss aber nicht, mit Hilfe der Identitäts-Infrastruktur gemacht werden. Zum Abschluss des Vorganges, generiert der Server einen geheimen Token, der mit der Bitte um Speicherung über die Identitäts-Schicht ausgetauscht wird. Speichert der Identitäts-Manager diesen Token in der Identität des Nutzers, so ist die Registrierung abgeschlossen. Je nach Implementierung und Einstellung des Sicherheitsdienstes ist zur Speicherung eine Interaktion mit dem Nutzer notwendig.

Authentifizierung eines registrierten Nutzers Wendet sich ein Nutzer erneut an den Server, kann ihn dieser nun automatisch authentifizieren. Hierzu muss der Server zunächst die Identität des Benutzers feststellen. Dazu sendet er einen anonymen `IdentityRequest` an den Manager des Nutzers. Dies bedeutet, dass der Server in seiner Anfrage den Identitäts-Bezeichner des Anwenders nicht angibt. Der Identitäts-Manager des Nutzers, generiert nach Rücksprache mit dem Sicherheitsdienst eine `IdentityAnswer`, deren enthaltene Identität an den Server weitergereicht wird. Dieser besitzt nun die angebliche Identität des Kommunikationspartners.

Da der Server jetzt eine Identitätsbehauptung besitzt, kann er mit deren Authentifizierung beginnen. Er stellt dazu eine gezielte Frage nach dem von ihm beim Nutzer gespeicherten Token. Diese Anfrage wird durch ein `AttributeRequest` realisiert. Hat der Nutzer die Herausgabe an den Server durch den Sicherheitsdienst erlaubt, wird ihm der geheime Token in einer Antwort-Identität übermittelt. Stimmt der übertragene Token mit dem Referenztoken beim Server überein, so ist die Identität authentifiziert.

Durch den Einsatz von kryptographischen Verfahren, kann das Sicherheitsniveau weiter erhöht werden. Dazu zählen zum Beispiel die verschlüsselte Übertragung der Daten oder das digitale Unterschreiben des Tokens.

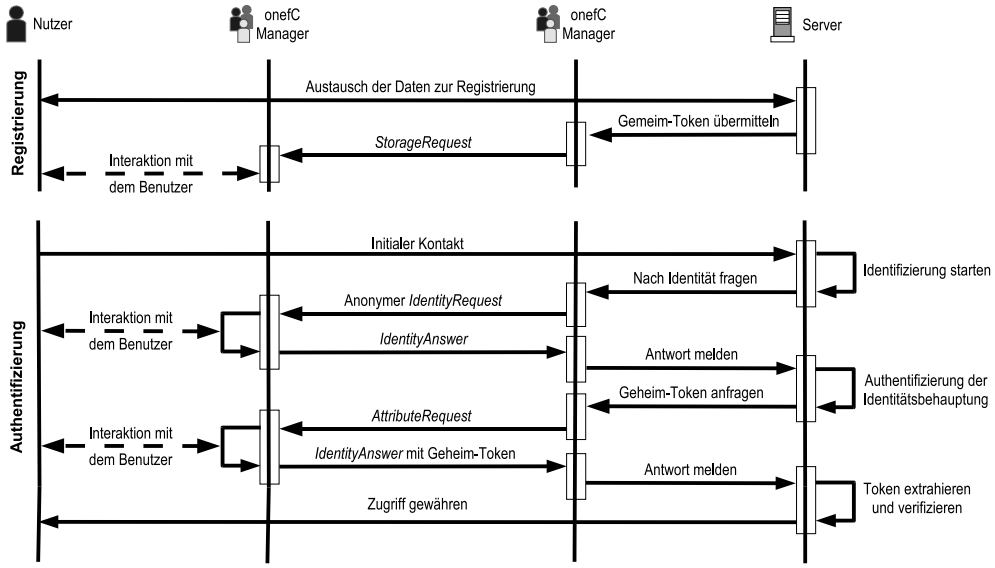

Abbildung 6.1: Authentifizierung als EF-Diagramm

6.2 Automatisches Ausfüllen von Formularen

Szenario-Beschreibung

Bei der Nutzung des World-Wide-Web tritt die Aufgabe des Ausfüllens eines Formulars immer wieder auf. Beim elektronischen Einkauf, bei Registrierungsvorgängen oder bei Umfragen im Internet müssen oftmals Daten in Formulare eingetragen werden.

Dieses Szenario ist im Kontext des E-Commerce angesiedelt. Ein Web-Shop möchte die Kaufabwicklung teilautomatisieren. Dazu sollen die Formulare mit den Daten des Nutzers vorbelegt werden. Dieser muss dann lediglich an den Stellen, die nicht mit den Standardwerten gefüllt sein sollen, Änderungen vornehmen. Die Identifizierung des Nutzers soll dabei schon bei Betreten des Shops geschehen.

Lösung mit dem onefC-Identitäts-Manager

Um den Online-Shop zu betreten, veranlasst der Nutzer seinen Browser, die zugehörige URL zu laden. Dazu wird vom Browser ein `HTTP-Request` an den Web-Server des Betreibers gesendet. Bevor dieser die Antwortseite generiert und an den Nutzer sendet, führt der Webserver eine Identifizierung des Nutzers durch. Dazu sendet er einen anonymen `IdentityRequest` an den Manager des Anwenders. Auf die Identitäts-Anfrage wird unter Einbeziehung des Sicherheitsdienstes ein Identitäts-Baum des Nutzers ausgewählt. Mit diesem als Datenbasis wird dann die `IdentityAnswer` generiert und zurück zum Web-Server übertragen. Speichert der Server die enthaltene Identität in einer Web-Session, bleibt diese während des gesamten Besuchs des Nutzers verfügbar. Somit kann der Server jederzeit auf die in ihr enthaltenen Daten zugreifen.

Ruft der Nutzer die Formularseite auf, so entnimmt der Server die Identitäts-Referenz und kann gezielt Daten über die Identitäts-Infrastruktur erfragen. In diesem Szenario erfragt der Web-Server durch einen `AttributeRequest` alle Attribute, die im E-Commerce Kontext stehen und durch das VCARD Schema definiert sind. Die durch die entsprechende `IdentityAnswer` übermittelten Attribute werden vom Web-Server extrahiert. Diejenigen Einträge, die einem Inhalt eines Feldes entsprechen, kann er nun als Vorgaben dem Formular hinzufügen und dieses dann an den Nutzer übertragen.

6.3 Wiedererkennen über Anwendungen hinweg

Szenario-Beschreibung

Eine interessante Möglichkeit, die sich durch den Einsatz einer einheitlichen Identitäts-Infrastruktur ergibt, ist das Wiedererkennen von Kommunikationspartnern. In diesem Szenario sollen die in unterschiedlichen Anwendungen gemachten Kontakte darauf hin überprüft werden, ob der Nutzer sein Gegenüber bereits aus einem anderen Kontext kennt. Dabei sollen keine Änderungen an den Anwendungen, die den Identitäts-Manager nutzen, gemacht werden müssen.

Lösung mit dem onefC-Identitäts-Manager

Um diese Aufgabe zu erfüllen, kann man sich eine Anwendung vorstellen, die sich beim Identitäts-Manager wie alle anderen Anwendungen registriert (siehe Abbildung 6.3).

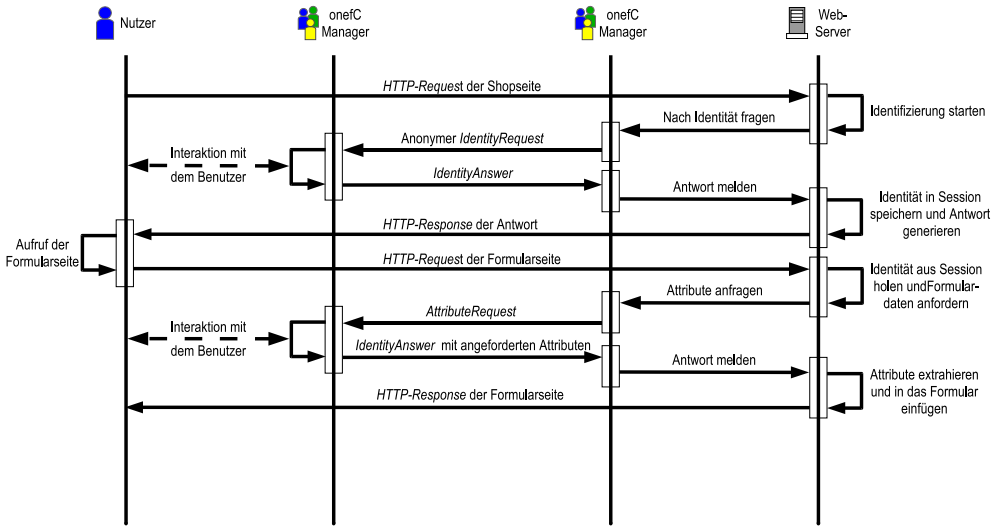

Abbildung 6.2: Automatisches Formularausfüllen als EF-Diagramm

Da die Anwendungen nach dem Prinzip des Beobachters über den Eingang von Nachrichten informiert werden, stehen ihnen automatisch alle eingehenden Nachrichten zur Verfügung. Die Anwendung zur Wiedererkennung von bereits bekannten Kommunikationspartnern, müsste lediglich die erhaltenen `IdentityAnswers` auswerten. Auf Grund des standardisierten Aufbaus kann das Programm auch ohne Verständnis der enthaltenen Attribute die für seine Aufgabe relevanten Daten ermitteln. Dies sind der Bezeichner der Identität und der Kontext, in dem die Übertragung stattfindet. Aus diesen Daten kann dann eine Datenbank aufgebaut werden, mit deren Hilfe die Wiedererkennung möglich ist. Dabei dienen die Identitäts-Bezeichner als Schlüssel. Ist ein Bezeichner in der Datenbank vorhanden, werden die bisherigen Kontexte angezeigt und der aktuelle gegebenenfalls hinzugefügt. Man könnte sich zusätzlich noch bestimmte Attribute wie den Nickname oder ähnliches anzeigen lassen.

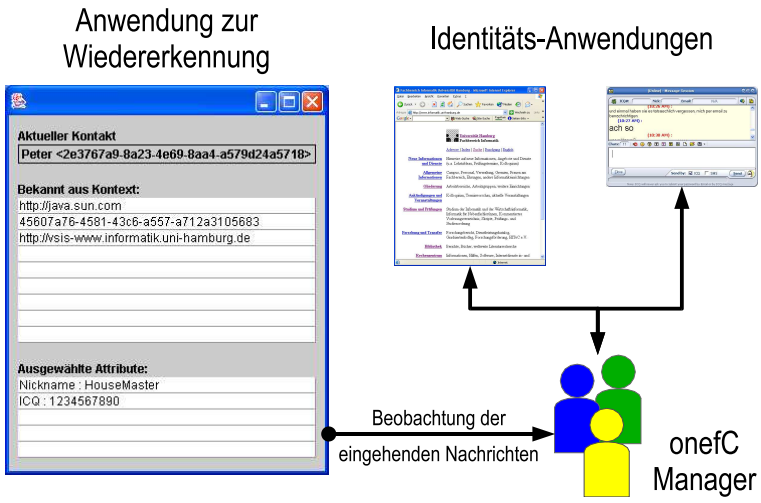

Abbildung 6.3: Mögliche Anbindung eines Programms zur Wiedererkennung

Kapitel 7

Zusammenfassung und Ausblick

Dieses Kapitel fasst die Ergebnisse dieser Arbeit über digitale Identitäten und Identitäts-Management zusammen und gibt einen Ausblick auf weitere Entwicklungsmöglichkeiten.

7.1 Ergebnisse der Arbeit

Die Betrachtung des Begriffs der Identität unter dem Gesichtspunkt des zwischenmenschlichen Verständnisses hat ergeben, dass er mehrere Aspekte abdeckt. Dazu gehört das für eine Wiedererkennung notwendige eindeutige Identifizieren von Objekten und Personen. Zudem wird mit der Identität auch ein inneres Konstrukt von Personen bezeichnet, dass subjektive und objektive Eigenschaften und Attribute umfasst. Dieser Aspekt der Identität führt zur Abgrenzung der eigenen Person und damit zur Ausbildung des einzelnen Individuums. Dieses Wahrnehmen des eigenen Selbst führt zum Begriff der Selbstdarstellung. Durch sie wird die private oder innere Dimension der Identität nach außen transportiert. Dabei können die offenbarten Attribute und Aussagen jedoch von denen des eigentlichen Selbst abweichen, um einen den eigenen Zielen entsprechenden Eindruck zu erzeugen. Dabei werden für die jeweils eingenommene Rolle spezielle Teil-Identitäten aktiviert. Der Aufbau dieser einzelnen Teile kann dabei sowohl auf Erfahrungen in der Realität oder im virtuellen Raum zurückgehen und baut ein sogenanntes Patchwork auf. Bewusste oder unbewusste Entscheidungen über die Mitgliedschaft in einer Gruppe und die Übernahme der durch sie repräsentierten Eigenschaften führt zur Integration sozialer Identitätsteile in das eigene Patchwork.

Das Identitäts-Management ist das technische Äquivalent zur Selbstdarstellung. Es bezeichnet Mechanismen zur Kontrolle und Beeinflussung der Weitergabe von Identitäts- oder Profildaten. Zu den Aufgaben des Identitäts-Managements zählen zudem der Schutz der persönlichen Identität, die Offenlegung der publizierten Daten sowie Authentifizierung und Pseudonymisierung. Es soll auch die Fragmentierung, also die räumlich Verteilung einzelner Teile der originären Identität, vermeiden.

Diese Erkenntnisse sind in das Konzept der vorliegenden Identitäts-Infrastruktur

eingeflossen. Dabei sind verschiedene Abbildungsmöglichkeiten der Identität untersucht worden. Für das in dieser Arbeit entwickelte Modell wurde eine Baumdarstellung gewählt, die als Abbild des Identitäts-Patchworks gesehen werden kann. Sie hat außerdem den Vorteil, dass durch Einführung eines Vererbungs- und Sichtbarkeitskonzepts die Wiederverwendung und die Anpassung der Attribute erleichtert wird. Das Attributmodell selbst ist ein Container-Modell, dass die in ihm enthaltenen Inhalte möglichst wenig einschränkt. Es definiert lediglich allgemeine Merkmale der Attribute und fügt sie in den Identitäts-Baum ein. Der entworfene Identitäts-Manager ist durch seinen Aufbau aus einzelnen Komponenten flexibel an die Bedürfnisse des jeweiligen Nutzers angepassbar. Vor allem durch die Verwendung eines individuellen Sicherheitsdienst kann er seine Schutzbedürfnisse definieren und durchsetzen. Durch die konsequente Protokollierung der Kommunikation der Management-Komponente wird versucht ein größtmögliches Maß an Offenheit zu erlangen. Die Umsetzung dieser Konzepte wurde abschließend in einem Prototypen realisiert.

Das Konzept erfüllt dabei wünschenswerte Eigenschaften. Es fokussiert auf einen universellen Einsatz, ist dezentral aufgebaut und konzentriert die persönlichen Daten beim Nutzer. Es erlaubt zudem einen fein gegliederten Datenschutz und legt die übermittelten Daten offen. Fügt man diese Eigenschaften in die Vergleichstabelle 3.3 ein, so ergeben sich folgende Übereinstimmungen und Unterschiede:

		Liberty Alliance	.NET Passport	XNS	onefC
Fokussierung auf	Universalität			◯	X
	World-Wide-Web	X	X		
Server	zentral	◯	X		
	dezentral			X	X
Datenhaltung	zentral		X		X
	dezentral	X		X	
Ort der Datenhaltung	beim Benutzer				X
	beim Anbieter	X	X	X	
Datenschutz	pauschal	X	X		
	fein gegliedert			X	X
Datenübermittlung	offengelegt			◯	X
	verborgen	X	X		

X := Erfüllt das Attribut
◯ := Erfüllt das Attribut teilweise

Tabelle 7.1: Einordnung der Eigenschaften der vorgestellten Architektur

Mit der onefC-Identitäts-Architekur ist ein System konzipiert, das sich in der digitalen Welt der Kommunikationsnetze so verhalten kann, wie es eine Person in der Realität macht. Attribute und Eigenschaften der digitalen Identität werden der Rolle und Situation entsprechend präsentiert. Dabei ist wie im alltäglichen Leben noch keine Angabe über den Wahrheitsgehalt der Aussage gemacht. Zunächst baut eine solche Beziehung auf dem Vertrauen der Partner untereinander auf. Verlangt eine Situation eine Bestätigung der gemachten Angaben, so müssen weitere Techniken in die Identitäts-Architektur eingebunden werden. Dazu können analog zur Realität Zertifikate und Ausweise eingesetzt werden. Diese bieten oftmals einen höheren Grad an Sicherheit, da sie beim ordnungsgemäßen Einsatz schwerer zu fälschen sind als ihre greifbaren Pendants. Die vorliegende Architektur kann also als Kern eines umfassenderen Konzeptes verstanden werden, der in den Situationen Einsatz finden kann, die keine oder geringe Garantien bezüglich der präsentierten Eigenschaften erfordern. Dazu zählen zum Beispiel die Bereiche des automatischen Ausfüllens von Formularen, Personalisierung von Netzinhalten oder das Wiedererkennen von Nutzern. Insgesamt wird mit dem vorliegenden Konzept ein elementarer Schritt auf dem Weg zum „Netizen“ oder Netzbürger beschritten.

7.2 Ausblick

Wie in Abschnitt 1.4 beschrieben wurde, sind die Ausformulierung und Implementierung der Mechanismen für Sicherheit und Vertrauen aus dieser Arbeit ausgegliedert worden. Diese sind für einen realen Einsatz der Architektur selbstverständlich zu konzipieren und umzusetzen. Auch die Entwicklung eines für die Identitäts-Kommunikation geeigneten Protokolls sollte noch betrieben werden. Weitere Ansätze zur Erweiterung der Fähigkeiten des Identitäts-Managers bestehen in der Entwicklung neuer Netzwerkadapter. Aber erst die Entwicklung von Anwendungen und Diensten, die die Identitäts-Infrastruktur einsetzen, kann ihr zu einer breiten Akzeptanz verhelfen.

Anhang A

XML-Schemata

A.1 Identitäts-Schema

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.christian-kunze.net/identity"
elementFormDefault="unqualified"
attributeFormDefault="unqualified">

<xsd:annotation>
<xsd:documentation xml:lang="de">
Identitaets Schema fuer das Onefc-Projekt.
Copyright 2002 Christian Philip Kunze.
All rights reserved.
</xsd:documentation>
</xsd:annotation>

<xsd:element name="IDENTITY" type="IdentityType"/>

<xsd:complexType name="IdentityType">
<xsd:sequence>
<xsd:element name="IDENTIFIER" type ="IdentifierType"/>
<xsd:element name="CONTEXTLIST" type="ContextListType"/>
<xsd:element name="ATTRIBUTELIST" type="AttributeListType"/>
<xsd:element name="SUBIDENTITYLIST" type="SubIdentityListType"/>
</xsd:sequence>
<xsd:attribute name="alias" type="xsd:string" use="optional"/>
<xsd:attribute name="ttl" type="xsd:date" use="optional"/>
</xsd:complexType>

<xsd:complexType name="ContextListType">
```

```
<xsd:sequence>
<xsd:element name="CONTEXT" type="ContextType"
             minOccurs="1" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ContextType">
<xsd:sequence>
<xsd:element name="IDENTIFIER" type ="IdentifierType"/>
</xsd:sequence>
<xsd:attribute name="alias" type="xsd:string" use="optional"/>
<xsd:attribute name="ttl" type="xsd:date" use="optional"/>
</xsd:complexType>

<xsd:complexType name="AttributeListType">
<xsd:all>
<xsd:element name="PROFILEATTRIBUTE" type="ProfileAttributeType"
             minOccurs="0"/>
<xsd:element name="FOREIGNPROFILEATTRIBUTE"
             type="ForeignProfileAttributeType" minOccurs="0"/>
<xsd:element name="SOCIALIDENTITYATTRIBUTE"
             type="SocialIdentityAttributeType" minOccurs="0"/>
</xsd:all>
</xsd:complexType>

<xsd:complexType name="SubIdentityListType">
<xsd:sequence>
<xsd:element name="IDENTITY" type="IdentityType" minOccurs="0"
             maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="IdentityAttributeType">
<xsd:sequence>
<xsd:element name="IDENTIFIER" type ="IdentifierType"/>
</xsd:sequence>
<xsd:attribute name="propagation" type="xsd:boolean" default="false"/>
<xsd:attribute name="ttl" type="xsd:date" use="optional"/>
</xsd:complexType>

<xsd:complexType name="ProfileAttributeType">
<xsd:complexContent>
<xsd:extension base="IdentityAttributeType">
```

```
<xsd:sequence>
<xsd:element name="NAME" type ="xsd:anyType" minOccurs="0"/>
<xsd:element name="CONTENT" type ="xsd:anyType" minOccurs="0"/>
</xsd:sequence>
<xsd:attribute name="language" type="xsd:string"/>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ForeignProfileAttributeType">
<xsd:complexContent>
<xsd:extension base="ProfileAttributeType">
<xsd:sequence>
<xsd:element name="ORIGINATOR" type="IdentifierType"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="SocialIdentityAttributeType">
<xsd:complexContent>
<xsd:extension base="IdentityAttributeType">
<xsd:attribute name="alias" type="xsd:string"/>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="IdentifierType">
<xsd:simpleContent>
<xsd:extension base="xsd:string">
<xsd:attribute name="type" type="xsd:string"/>
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
</xsd:schema>
```

A.2 Nachrichten-Schema

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:id="http://www.christian-kunze.net/identity"
targetNamespace="http://www.christian-kunze.net/messages"
elementFormDefault="unqualified"
attributeFormDefault="unqualified">

<xsd:import namespace="http://www.christian-kunze.net/identity"
     schemaLocation=
     "http://vsis-www.informatik.uni-hamburg.de/~6kunze/DA/identity.xsd"/>

<xsd:element name="IDETITYMANAGERMESSAGE" type="MessageType"/>

<xsd:complexType name="MessageType">
<xsd:sequence>
<xsd:element name="HEADER" type="HeaderType"/>
<xsd:element name="BODY" type="BodyType"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="HeaderType">
<xsd:sequence>
<xsd:element name="FROM" type="UniqueType"/>
<xsd:element name="TO" type="UniqueType" minOccurs="0"/>
<xsd:element name="CONTEXT" type="id:ContextType"/>
<xsd:element name="REFERENCE" type="UniqueType"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="BodyType">
<xsd:sequence>
<xsd:choice>
<xsd:element name="IDENTITYREQUEST" type="IdentityRequestType"/>
<xsd:element name="ATTRIBUTEREQUEST" type="AtributeRequestType"/>
<xsd:element name="IDENTITYANSWER" type="IdentityAnswereType"/>
</xsd:choice>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="UniqueType">
<xsd:sequence>
<xsd:element name="IDENTIFIER" type="id:IdentifierType"/>
```

```
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="IdentityRequestType">
<xsd:sequence>
<xsd:choice>
<xsd:element name="FOREIGNIDENTITY" type="UniqueType"/>
<xsd:element name="LOCALIDENTITY"/>
</xsd:choice>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AtributeRequestType">
<xsd:sequence>
<xsd:element name="ATTRIBUTELIST" type="id:AttributeListType"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="IdentityAnswereType">
<xsd:sequence>
<xsd:element name="IDENTITY" type="id:IdentityType"/>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[AH01] Abdelal, Rawi und Yoshiko M. Herrera: *treating identity as a variable: measuring the content, intensity, and contestation of identity*. Bericht, Harvard Business School, August 2001.

[BK00] Berthold, Oliver und Marit Köhntopp: *Identity Management Based On P3P*. In: *Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[BL94] Berners-Lee, Tim: *RFC 1630 - Universal Resource Identifiers in WWW*. Bericht, World Wide Web Consortium - Network Working Group, Juni 1994.

[BL98] Berners-Lee, Tim: *RFC 2396 - Uniform Resource Identifier (URI): Generic Syntax*. Bericht, World Wide Web Consortium - Network Working Group, August 1998.

[BL02] Berners-Lee, Tim: *Naming and Adressing: URIs, URLs, ...* Web-Seite, Juli 2002. *http://www.w3.org/Adressing/*, Abruf am 24.03.2003.

[Bor96] Borking, John: *Der Identity-Protector*. Web-Seite, 1996. *http://www.datenschutzzentrum.de/somak/somak96/sa96brok.htm*, Abruf am 31.07.2002.

[BRO89a] *BROCKHAUS ENZYKLOPÄDIE in vierundzwanzig Bänden: Zehnter Band HERR – IS*. F.A. Brockhaus, 1989.

[BRO89b] *BROCKHAUS ENZYKLOPÄDIE in vierundzwanzig Bänden: Zwölfter Band Kir – LAG*. F.A. Brockhaus, 1989.

[BZL03] Baier, Toby, Christian Zirpins und Winfried Lamersdorf: *Digital Identity: How to be Someone on the Net*. In: *Proceedings of the IADIS International Conference of e-Society*, 2003. Veröffentlichung zum Juni 2003.

[Cap01] Cap, Clemens H.: *Digital Identity and it's Implications for Electronic Government*. Bericht, Universität Rostock, Juni 2001.

[CDK01] Coulouris, George, Jean Dollimore und Tim Kindberg: *Distributed Systems Concepts and Design*. Addison-Wesley, 3 Auflage, 2001.

[Cla99] Clarke, Roger: *Identified, Anonymous and Pseudonymous Transactions: The Spectrum of Choice*. Web-Seite, April 1999. *http://www.anu.edu.au/people/Roger.Clarke/DV/UIPP99.html*, Abruf am 31.07.2002.

[CMB02] Casassa Mont, Marco und Richard Brown: *Active Digital Credentials: Dynamic Provision of Up-to-Date Identity Information*. Bericht, HP - Trust, Security and Privacy, 2002.

[CMBG+02] Casassa Mont, Marco, Pete Bramhall, Mikey Gittler, Joe Pato und Owen Rees: *Identity Management: a Key e-Business Enabler*. Bericht, HP - Trusted E-Services Laboratory, Juni 2002.

[Deb96] Debatin, Bernhard: *Elektronische Öffentlichkeiten. Über Informationsselektion und Identität in virtuellen Gemeinschaften*. Web-Seite, April 1996. *http://www.uni-leipzig.de/~debatin/english/Articles/Fiff.htm*, Abruf am 22.06.2002.

[Dör99] Döring, Nicola: *Sozialpsychologie des Internet*. Hogrefe, 1999.

[Dör00] Döring, Nicola: *Identität + Internet = Virtuelle Identität?* In: *forum medienethik*, Band 2, Seiten 65–75. KoPäd Verlag, 2000.

[Don96] Donath, Judith S.: *Identity and deception in virtual community*. Bericht, MIT Media Lab, Juli 1996.

[DT02] D-TRUST: *Unverzichtbar in der Netzwelt: Die digitale Identität durch PKI*. Web-Seite, Juli 2002. *http://www.d-trust.de/internet/content/003.html*, Abruf am 22.06.2002.

[Dud96] Dudenreadktion (Herausgeber): *Duden, Rechtschreibung der deutschen Sprache*, Band 21. 1996.

[ECS94] Eastlake, Donald E., Stephen D. Crocker und Jeffrey I. Schiller: *RFC 1750 - Randomness Recommendations for Security*. Bericht, World Wide Web Consortium - Network Working Group, Dezember 1994.

[Fla98] Flanagan, David: *Java in a Nutshell - Deutsche Ausgabe für Java 1.1*. O'Reilly, 2 Auflage, 1998.

[Fuc02] Fuchs, T.: *Der Begriff der Person in der Psychiatrie*. Der Nervenarzt, 73(3):239–246, 2002.

[GHJV96] Gamma, Erich, Richard Helm, Ralph Johnson und John Vilissides: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Addison Wesley Verlag, 1996.

[Gol99] Goland, Y.: *RFC 2518 - HTTP Extensions for Distributed Authoring*. Bericht, World Wide Web Consortium - Network Working Group, Februar 1999.

[Hen76] Henrisch, Dieter: *Identität und Objektivität: eine Untersuchung über Kants transzendentale Deduktion*. In: *Sitzungsberichte der Heidelberger Akademie der Wissenschaften – Philosophisch-Historische Klasse*, Band 1, Seite 54ff. Winter, 1976.

[HW03] Hodges, Jeff und Tom Wason: *Liberty Architecture Overview*. Bericht 1.1, Liberty Alliance Project, Januar 2003.

[JG00] Jendricke, Uwe und Daniela Gerd tom Markotten: *Usability meets Security - The Identity-Manager as your Personal Security Assistant for the Internet*. In: *Proceedings of the 16th Annual Computer Security Applications Conference*, Dezember 2000.

[JG01a] Jendricke, Uwe und Daniela Gerd tom Markotten: *Identitätsmanagement: Einheiten und Systemarchitektur*. In: Fox, Dirk, Marit Köhntopp und Andreas Pfitzmann (Herausgeber): *Verlässliche IT-Systeme - Sicherheit in komplexen Infrastrukturen*, Seiten 77–85. Vieweg, Wiesbaden, September 2001.

[JG01b] Jendricke, Uwe und Daniela Gerd tom Markotten: *Identitätsmanagement im E-Commerce*. it+ti Informationstechnik und Technische Informatik, 43(5):236–245, Oktober 2001.

[JGM01] Jendricke, Uwe, Daniela Gerd tom Markotten und Günter Müller: *Benutzbare Sicherheit – Der Identitätsmanager als universelles Sicherheitswerkzeug*, Kapitel 7, Seiten 135–146. Springer-Verlag Berlin, Mai 2001.

[JKZ02] Jendricke, Uwe, Michael Kreutzer und Alf Zugenmaier: *Mobile Identity Management*. Bericht 178, Institut für Informatik, Universität Freiburg, Oktober 2002. Workshop on Security in Ubiquitous Computing, UBICOMP 2002.

[JP00] Jakobsson, Mikael und Victoria L. Popdan: *How we became net friends, and what we learned from it*. Bericht, 2000.

[Köh00] Köhntopp, Marit: *Identitätsmanagement – Anforderungen aus Nutzersicht*. Bericht, NRW-Forschungsverbund Datensicherheit, Juli 2000.

[Koc02] KOCH, MICHAEL: *Global Identity Management to Boost Personalization*. In: SCHUBERT, PETRA und UWE LEIMSTOLL (Herausgeber): *Proceedings of the Ninth Research Symposium on Emerging Electronic Markets*, Seiten 137–147. Institute for Business Economics, September 2002.

[KR00] KORMANN, DAVID P. und AVIEL D. RUBIN: *Risks of the Passport Single Signon Protocol*. Computer Networks, 33:51–58, 2000. *http://avirubin.com/passport.html*, Abruf am 28.01.2003.

[Kra96] KRAMER, DOUGLAS: *The Java Platform*. Bericht, Sun Microsystems, Mai 1996.

[KW01] KOCH, MICHAEL und WOLFGANG WÖRNDL: *Community Support and Identity Management*. Bericht, Technische Universität München, September 2001.

[KW02] KUSCHKE, MICHAEL und LUDGER WÖLFEL: *Generalschlüssel - Microsoft Passport und Konkurrenten*. iX - Magazin für Professionelle Informationstechnik, (2):96–98, Februar 2002.

[Lip98] LIPINSKI, KLAUS (Herausgeber): *Lexikon TCP/IP Internetworking*. International Thompson Publishing, 1998.

[Mic02a] MICROSOFT CORPORATION: *.NET Passport Overview*. Web-Seite, 2002. *http://www.microsoft.com/netservices/passport/overview.asp*, Abruf am 28.01.2003.

[Mic02b] MICROSOFT CORPORATION: *.net Passport: Review Guide*. Bericht, März 2002.

[Min02] MINTERT, STEFAN (Herausgeber): *XML & Co.* ADDISON-WESLEY, 2002.

[Mit84] MITTELSTRASS, JÜRGEN: *Enzyklopädie Philosophie und Wissenschaftstheorie 2*. Wissenschaftsverlag, 1984.

[Moa97] MOATS, RAYAN: *RFC 2141 - URN Syntax*. Bericht, World Wide Web Consortium - Network Working Group, Mai 1997.

[One03] ONENAME CORPORATION: *Overview of eXtensible Name Service*. Web-Seite, 2003. *http://www.onename/pages/ps_axns.htmlrl*, Abruf am 29.01.2003.

[OPE97] *DCE 1.1: Remote Procedure Call*. Bericht, Open Group, Oktober 1997.

[Res98] RESCH, FRANZ: *Zur präpsychotischen Persönlichkeitsentwicklung in der Adoleszenz*. Psychotherapeut, 43(2):111–116, 1998.

[Sig01] *Gesetz über Rahmenbedingungen für elektronische Signaturen (Signaturgesetz – SigG)*, Mai 2001. *http://www.bmwi.de/Homepage/download/infogesellschaft/Signaturgesetz1.pdf*, Abruf am 10.02.2003.

[Sul96] SULER, JOHN: *Identity Management in Cyberspace*. Web-Seite, 1996. *http://www.rider.edu/users/suler/psycyber/identitymanage.html*, Abruf am 06.10.2002.

[Tan97] TANENBAUM, ANDREW S.: *Computernetzwerke*. Prentice Hall, 3 Auflage, 1997.

[TS02] TANENBAUM, ANDREW S. und MAARTEN VAN STEEN: *Distributed Systems Principles and Paradigms*. Prentice Hall, 2002.

[Tur99] TURKLE, SHERRY: *Leben im Netz: Identität in Zeiten des Internet*. Rohwolt Taschenbuch Verlag, Juni 1999.

[Wai00] WAISMANN, FRIEDRICH: *Über den Begriff der Identität*. Web-Seite, 2000. *http://www.mauthner-gesellschaft.de/mauthner/tex/wais2.htm*, Abruf am 22.06.2002.

[XNS02a] XNS PUBLIC TRUST ORGANIZATION: *From Name Service to Identity Service: How XNS Builds on the DNS Model*. Bericht, Juli 2002.

[XNS02b] XNS PUBLIC TRUST ORGANIZATION: *The identity Web: key Concepts of XNS Architecture*. Bericht, Juli 2002.

[XNS02c] XNS PUBLIC TRUST ORGANIZATION: *An Introduction to XNS, the eXtensible Name Service*. Bericht, Januar 2002.

[XNS02d] XNS PUBLIC TRUST ORGANIZATION: *XNS Service Models*. Bericht, Juli 2002.

Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

Außerdem erkläre ich, dass ich mit der Einstellung dieser Diplomarbeit in den Bestand der Bibliothek des Fachbereiches Informatik der Universität Hamburg einverstanden bin.

Hamburg, den

Christian Kunze