HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

Diplomarbeit

Herr
Marcus Pemmann

Konzeption und prototypische Implementierung der Integration einer bestehenden JEE Geschäftsanwendung in ticketbasierte Single Sign-On Systeme

2010

Fakultät **Informationstechnik & Elektrotechnik**

Diplomarbeit

Konzeption und prototypische Implementierung der Integration einer bestehenden JEE Geschäftsanwendung in ticketbasierte Single Sign-On Systeme

Autor:
Marcus Pemmann

Studiengang:
Multimediatechnik

Seminargruppe:
MK05w2

Erstprüfer:
Prof. Dr.-Ing. Frank Zimmer

Zweitprüfer:
Dipl.-Inf. Mike Richter

Mittweida, Januar 2010

Bibliographische Beschreibung

Pemmann, Marcus:
Konzeption und prototypische Implementierung der Integration einer bestehenden JEE Geschäftsanwendung in ticketbasierte Single Sign-On Systeme. - 2010. - 78 S.
Mittweida, Hochschule Mittweida, Fakultät Informationstechnik & Elektrotechnik, Diplomarbeit, 2010

Referat

Um den gezielten Einsatz von Ressourcen zu optimieren, bündelt man wiederholt auftretende Prozesse, definiert Schnittstellen und nutzt Standards. Vorhandenes Potenzial effektiv einzusetzen bedeutet auch, Abläufe zu reduzieren, die stets nach demselben Schema erfolgen und nicht dem eigentlichen Nutzen dienen. Die Eingabe von Informationen zur Identifikation von Benutzern fällt in diese Kategorie.

Im Rahmen dieser Arbeit wird anhand einer vorhandenen Geschäftsanwendung demonstriert, wie die Integration in Systeme zur Einmalanmeldung mit Hilfe von Schnittstellen in Java umgesetzt werden kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface (Programmierschnittstelle)
BLS	Business Logic Server der DELTA proveris AG
CAS	Central Authentication Service
CTO	CommandTO, Kommandotransportobjekt
DB	Datenbank, engl. Database
HTTP(S)	Hypertext Transfer Protocol (Secure)
ID	Identifikator, engl. Identifier
JAAS	Java Authentication and Authorization Service
Java EE / JEE	Java Platform Enterprise Edition
JOSSO	Java Open Single Sign-On
LDAP	Lightweight Directory Access Protocol
SAML	Security Assertion Markup Language
SDK	Software Development Kit
SID	Session-ID, Identifikator einer Sitzung
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
SSO	Single Sign-On
URL	Uniform Resource Locator
WAR	Web Application Archive
WS	Web Service
XML	Extensible Markup Language

1 Einleitung

1.1 Hintergrund

Bei einer steigenden Anzahl von Benutzerkonten entsteht das **Problem**, sich **mehrere Login-Passwort-Kombinationen** merken zu müssen. Die scheinbare Abhilfe, überall die gleiche Kombination oder wenigstens das gleiche Passwort zu verwenden – was nicht immer möglich ist, da registrierbedürftige Anwendungen oder Dienste verschiedene Vorgaben diesbezüglich haben – bringt ein Sicherheitsproblem mit sich. Falls das Kennwort geknackt werden kann, hat der Angreifer im schlimmsten Fall Zugriff auf alle entsprechenden Accounts. Alternativ wählt man die zu vergebenden Kennwörter so, dass sie leicht zu merken sind oder schreibt sie, vielleicht sogar gut sichtbar, auf Klebezettel – und erzielt einen ähnlichen Effekt.

*"Every **Single** time you want to do something,*
*you are going to have to **Sign-On**!"*

- Java One Conference 2007 [oso] S. 5

Als **Lösung**, die auch den Sicherheitsaspekt berücksichtigt, kann **Single Sign-On** (SSO) angesehen werden. Der Begriff wird im Allgemeinen mit „Einmalanmeldung" übersetzt; der Umfang einer SSO-Lösung wird unterschiedlich interpretiert. Stark vereinfacht dargestellt, verbirgt sich dahinter die Anmeldung eines Benutzers an einem System, etwa einem PC oder einer Webanwendung, mittels Nutzername und Passwort, woraufhin alle folgenden Anmeldevorgänge innerhalb der gestarteten Sitzung automatisiert ablaufen.

Der **Gewinn** liegt klar auf der Hand: Durch die nunmehr einzige Nutzername-Passwort-Kombination fallen die vielen unsicheren Kennwörter oder Klebezettel weg. Das eine verbleibende Passwort kann entsprechend komplex gewählt werden, was

durch Länge und vielfältige Verwendung von Sonderzeichen erreicht wird. Gerade beim Einsatz von SSO im Web[1] reduziert sich der Aufwand für die Wartung von Zugangsdaten wie etwa die Bearbeitung von Anfragen zum Zurücksetzen des Kennworts. Das bedeutet Einsparung von Zeit und Kosten. Gleiches geschieht natürlich auch auf der Nutzerseite durch den Wegfall der interaktiven Anmeldevorgänge. Hinzu kommt ein Sicherheitsgewinn, da das Kennwort nur noch ein einziges Mal übertragen werden muss und so Phishing[2]-Attacken erschwert werden.

Selbstverständlich sollen auch die **Risiken** nicht unbeachtet gelassen werden. Falls es einem Angreifer gelingen sollte, das Kennwort auszuspähen, hätte er Zugang zu allen Accounts des Nutzers, die in das SSO-System integriert sind. Aus diesem Grund ist an die hohe Komplexität des Kennwortes zu appellieren, sowie an eine unbedingt zu verschlüsselnde Übertragung von sensiblen Nutzerdaten. Weiterhin ist zu bedenken, dass die Verfügbarkeit der Dienste und Anwendungen direkt vom SSO-System abhängt, was wiederum Überlegungen hinsichtlich einer redundanten Infrastruktur erforderlich macht, sofern diese nicht ohnehin bereits vorhanden ist. Ein mögliches Risiko besteht auch darin, Kunden zu verlieren, die das eingesetzte System aufgrund von Sicherheitsbedenken nicht akzeptieren.

Die **DELTA proveris AG**, mit Sitz in Limbach-Oberfrohna und Teil der DELTA BARTH Unternehmensgruppe, entwickelt professionelle Softwarelösungen für Leasinganbieter und Fuhrparkbetreiber. Das Kernprodukt **Leasman** stellt komplette Organisationsabläufe im Fahrzeugleasing dar [lma].

Leasman wurde ursprünglich als 2-Schicht-Architektur entwickelt. Durch die Auslagerung der Geschäftslogik in eine serviceorientierte Komponente, den **Business Logic Server** (BLS), wurde eine Mittelschicht zwischen Datenbank und Präsentationsschicht eingeführt. Der Zugang zum BLS erfolgt über standardisierte Schnittstellen. Dadurch steht die Funktionalität neben Leasman auch anderen Client-Anwendungen zur Verfügung.

1 „Web“ wird im Rahmen dieser Arbeit als Alias für „World Wide Web“ verwendet

2 Abfischen vertraulicher Daten

1.2 Motivation

Effektivität und Potenzial – zwei Bestandteile des Leitbildes der DELTA proveris AG. Um den gezielten Einsatz von Ressourcen und rationelles Handeln weiter zu optimieren, bündelt man wiederholt auftretende Prozesse, definiert Schnittstellen und nutzt Standards. Vorhandenes Potenzial effektiv einzusetzen bedeutet auch, Abläufe zu eliminieren oder auf ein Mindestmaß an Zeitaufwand zu reduzieren, die zwar unumgänglich sind, jedoch stets nach demselben Schema erfolgen und nicht dem eigentlichen Nutzen dienen.

Die Eingabe von Informationen zur Identifikation von Benutzern gehört zweifellos in diese Kategorie. Im Bestreben, an dieser Stelle Zeit und damit Kosten einzusparen, wurde die Aufgabe gestellt, eine mögliche Integration des Leasman BLS in Single Sign-On Systeme zu untersuchen. Diese bieten eine Vielzahl von Schnittstellen für gängige Software. Da die Integration von Anwendungen jedoch sehr komplex sein kann, reicht es nicht aus, vorhandene Systemkomponenten anzupassen. Zur Entwicklung geeigneter Bindeglieder sind Programmierschnittstellen nötig, welche die SSO-Lösung zur Verfügung stellen muss.

Als **Ziel** dieser Arbeit soll ein Prototyp der Integration entwickelt und getestet werden. Für den Fall, dass sich mehrere Client-Anwendungen am Business Logic Server anmelden, sollen diese Anmeldevorgänge automatisiert über die SSO-Umgebung erfolgen. Unter den Aspekten der Portierbarkeit, Skalierbarkeit und Erweiterbarkeit ist es dabei eine wichtige Aufgabe, geeignete Schnittstellen zu definieren, welche die Anbindung weiterer Anwendungen möglich machen. Es soll gezeigt werden, dass und wie die Single Sign-On Integration unter diesen Umständen umsetzbar ist.

Das Ziel ist kein neues Single Sign-On System – davon gibt es bereits eine hinreichend große Anzahl – sondern ein Prototyp, der die Integration des BLS in vorhandene SSO-Systeme ermöglicht. Die Implementierung wird prototypisch für ausgewählte Systeme getestet.

1.3 Gliederung

Zu Beginn des 2. Kapitels wird eine **Basis** für die Auseinandersetzung mit dem Thema geschaffen, indem Grundlagen behandelt werden, auf denen Single Sign-On Systeme aufbauen. In Abschnitt 2.2 wird die Geschäftsanwendung Leasman BLS charakterisiert.

Daran anknüpfend werden in Kapitel 3 vorhandene **SSO**-Lösungsansätze und konkrete **Systeme** im Umfeld der Java Platform Enterprise Edition beleuchtet und hinsichtlich der Anwendungsintegration geprüft.

Es folgt im 4. Kapitel ein **Entwurf** zur Integration auf Basis der Erkenntnisse über SSO-Systeme.

Anschließend wird in Kapitel 5 die prototypische **Implementierung** beschrieben.

Die Arbeit schließt im 6. Kapitel mit einem **Fazit**, welches die Ergebnisse zusammenfasst und gleichzeitig einen Ausblick bietet.

2 Grundlagen

Im Kontext des Single Sign-On stammt das Vokabular aus den Bereichen IT-Sicherheit, Nutzerverwaltung und heterogenen Netzen. Die Kenntnis von Begriffen und Vorgängen im IT-Bereich, speziell zu Netzwerken und Protokollen, sowie Ausdrücken aus der Java-Welt wird vorausgesetzt. Zur weiteren Vertiefung können die Quellen im Literaturverzeichnis genutzt werden.

Nachfolgend werden häufig verwendete Begriffe eingeführt. Die Beschreibungen können von denen anderer Quellen abweichen.

2.1 Informationstechnische Grundlagen

2.1.1 Authentisierung und Autorisierung

Bei der **Authentisierung** wird die Identität eines Benutzers anhand von Berechtigungsnachweisen überprüft. Mitunter findet eine Abgrenzung zum Begriff Authentifizierung statt. Damit ist der Vorgang aus Sicht der Instanz gemeint, welche den Identitätsnachweis entgegen nimmt und prüft, wohingegen die Authentisierung aus der Nutzersicht erfolgt.

In der Praxis werden unterschiedliche **Formen** der Authentisierung verwendet, teilweise werden diese auch kombiniert. Ein Weg führt über den **Besitz** eines Schlüssels oder einer Chipkarte. Ein Weiterer nutzt das **Wissen**, das nur die Person haben sollte, die sich anmelden möchte, z.B. ein Passwort oder eine PIN. Als dritter Weg werden **Merkmale** genutzt, die jede Person eindeutig identifizieren, etwa der Fingerabdruck oder die Netzhaut des Auges. In der vorliegenden Arbeit wird auf diese biometrischen Verfahren sowie auf weitere Hardware-basierte Formen wie Chipkarten oder USB-Tokens nicht näher eingegangen.

Als Primärauthentisierung wird die erstmalige – und im Rahmen von SSO auch einmalige – Eingabe der Logindaten bezeichnet. Durch die Authentisierung gelangt der Nutzer zu einer Authentizität, einer als gültig bestätigten digitalen Identität.

Die folgende Tabelle listet verschiedene Authentisierungsformen (Authentication Schemes) auf, wie sie in Single Sign-On Systemen eingesetzt werden:

Simple, Basic	Nutzername und Passwort aus Datei
Bind	Über Daten aus Data Store (LDAP)
Strong	Mit X.509 Zertifikat
Remember-Me	Tickets mit Langzeitauthentifizierung

Bei der einfachsten Form der Authentisierung, der *simple-* oder *basic-authentication*, wird die Eingabe von Loginname und Passwort mit den Daten aus einer Datei verglichen. Die Daten liegen im Klartext oder zum grundlegenden Schutz gegen Missbrauch verschlüsselt vor. Stimmen die Eingaben mit den Vergleichsdaten überein, ist die Anmeldung erfolgreich. Eine *bind-authentication* verwendet als Speicher für die Authentisierungsinformationen einen Verzeichnisdienst. Dieser greift auf die hierarchische Struktur einer Organisation zu und verifiziert die eingegebenen Daten anhand der Einträge im *Data Store*. Als *strong* wird eine Authentisierung bezeichnet, wenn eine Verschlüsselung mit Hilfe von Zertifikaten geschieht. Eine besondere Form, die vor allem bei Webanwendungen verwendet wird, ist *remember-me*. Über Cookies im Webbrowser, die eine lange Laufzeit haben, kann ein Nutzer bei der selben Anwendung immer wieder authentifiziert werden, ohne Eingaben machen zu müssen.

Eine „Methode zur Authentifizierung von Anwendern gegenüber Programmen und Hosts“ [ldp03] ist **Kerberos**. Dabei werden sogenannte Granting Tickets mit einem Schlüssel und einer Gültigkeitsdauer an Stelle von Passwörtern übertragen. Da Kerberos auch als Lösung für Single Sign-On gilt, wird im Kapitel 3.1.3 „Kerberos“ näher darauf eingegangen. Ein weiteres Verfahren zur Authentifizierung ist NTLM

(Windows NT LAN Manager), dem allerdings aufgrund von Sicherheitsbedenken Kerberos vorgezogen werden sollte.

Nach erfolgreicher Authentisierung kann die **Autorisierung** stattfinden. Zunächst muss ermittelt werden, wer die Person ist, die sich anmelden möchte. Anschließend kann überprüft werden, was diese Person darf. Der Nutzer erhält bestimmte Rechte zum Zugriff auf Dienste und Anwendungen. Das können z.B. Lese- oder Schreibrechte in einem Verzeichnis sein. Im Fall des Leasman stehen jedem Nutzer nach der Eingabe eines Bedienernamens mit zugehörigem Kennwort verschiedene Mandanten zur Auswahl, die jeweils mit unterschiedlichen Rechten verbunden sind.

Häufig wird für Vorgänge der Authentisierung und Autorisierung auch der Begriff *Identity and Access Management* verwendet.

Der hier zu entwickelnde Softwareprototyp soll vorrangig die Prozesse zur Authentisierung abbilden. Die Verwaltung und Behandlung von Rollen und Rechten verbleibt bei der Geschäftsanwendung, da diese mit ihrer Komplexität den Umfang der Arbeit übersteigt.

2.1.2 Credentials

Die Berechtigungsnachweise zur Authentisierung werden auch **Credentials** genannt und können beispielsweise Passwörter, Tickets, Zertifikate oder Schlüssel sein.

Richtlinien für gute **Passwörter** schreiben verschiedene Kriterien vor, sodass eine komplexe Kombination aus Buchstaben und Zahlen sowie Zeichen entstehen kann. Solch ein Passwort lässt sich nur schwer merken, gerade wenn es sich um eine Vielzahl von Kennwörtern handelt. Zur Erleichterung will sich die Einmalanmeldung auf *ein* „gutes Passwort“ bei der Primärauthentisierung beschränken und anschließend Credentials verwenden, die keine Kennworteingabe erfordern.

Ein weiterer Ansatz ist das **One Time Password** (OTP). Dabei wird jedes Passwort nur einmal verwendet, man muss sich keines merken. Die Passwörter werden entwe-

der über Listen ermittelt, wie sie von der TAN-Liste[3] beim PIN/TAN-Verfahren aus dem Online-Banking her bekannt sind, oder durch einen Passwortgenerator erzeugt.

Tickets sind „verschlüsselte Datenpakete [...], die kryptographische Informationen zur Authentizitätskontrolle der Kommunikationspartner enthalten." [krb] Sie eliminieren die mehrfache Eingabe von Nutzername und Passwort in SSO-Systemen und werden bei der ersten Anmeldung an einem zentralen SSO-Server oder dem Mitglied eines Verbundes aus Anwendungen, die sich gegenseitig vertrauen[4], erzeugt. Aus Sicherheitsgründen, um unerlaubte Änderungen am Ticket zu verhindern und das Risiko des Abfangens zu minimieren, sollten Tickets immer eine Gültigkeit besitzen und verschlüsselt übertragen werden. [sso]

Ein digitales **Zertifikat** wird von einer Certificate Authority (CA) ausgestellt und bestätigt die Identität eines Nutzers. Es handelt sich um eine Datenstruktur, die den Namen des Eigentümers, Schlüssel, Gültigkeit und weitere Merkmale enthalten kann. Für eine zertifikatsbasierte Authentifizierung ist eine Public Key Infrastructure (PKI) erforderlich. Bei PKI wird einem Benutzer ein Paar aus öffentlichem und privatem Schlüssel (public und private key) zugewiesen. Mit dem *public key* werden Nachrichten des Eigentümers verschlüsselt und können nur mit dem geheimen *private key* entschlüsselt werden.

2.1.3 Session

Verteilte Systeme bestehen aus Client- und Serverkomponenten. Für den Fall, dass ein Client Dienste des Servers in Anspruch nehmen will, muss eine Verbindung hergestellt werden. Die Verbindung allein benötigt noch keine Sitzung. Sollen jedoch Authentisierungsinformationen übertragen werden, um die Anfragen an den Server einem bestimmten Client zuordnen zu können, muss eine *Session* initiiert werden. Diese erhält als eindeutiges Identifikationsmerkmal eine Session-ID. Bei jedem Re-

3 TAN: Transaction Number

4 Circle of Trust: siehe Kapitel 3.1.1 „Zentral oder Verbund"

quest, bei dem der Client Daten vom Server anfordert, muss die ID mitgesendet werden. Ebenso ist sie für die Serverantwort erforderlich. Aus Sicherheitsgründen besitzen Sessions eine begrenzte Laufzeit. Die Sitzung kann jedoch auch vor Ablauf dieser Zeit durch den Nutzer beendet werden.

2.1.4 Verzeichnisdienste

Zu den grundlegenden Bestandteilen von Single Sign-On gehört die Verwaltung von Nutzerdaten. Als Nutzer wird im vorliegenden Kontext eine Person gesehen, die über einen Account verfügt und damit die Möglichkeit erhält, sich zur personalisierten Nutzung an einem Server zu authentisieren. Die personenbezogenen Informationen werden in einer Datenbank oder einem Verzeichnisdienst gehalten. Eine häufig verwendete Bezeichnung für den Nutzerdatenspeicher ist *Data Store*.

Verzeichnisdienste stellen Informationen in einem Netzwerk zur Verfügung. Sie bilden Daten in hierarchischen Strukturen ab. Der Leitfaden für die Migration von Software des Bundesinnenministeriums stellt folgendes fest:

> *„Verzeichnisdienste eignen sich insbesondere für den schnellen lesenden Zugriff auf hierarchisch strukturierte Daten, die nicht regelmäßig geändert werden. […]*
> *Mit der Einführung eines Verzeichnisdienstes ist es möglich, die Benutzerkonten und die dazugehörigen Berechtigungen zentral im Verzeichnisdienst zu speichern und alle Systeme darauf zugreifen zu lassen." [mig] S.187*

LDAP

Eine Möglichkeit des Zugriffes auf einen Verzeichnisdienst bietet das Lightweight Directory Access Protocol (LDAP), das am häufigsten eingesetzte Verzeichnisdienstprotokoll. LDAP ist ein Standard der Internet Engineering Task Force (IETF) und in Version 3 im Request For Comment (RFC) Nr. 4511 dokumentiert[5].

5 siehe http://tools.ietf.org/html/rfc4511

Bezüglich Single Sign-On ist in dem Buch *LDAP verstehen – OpenLDAP einsetzen* vermerkt: „Die Benutzer- und Diensteverwaltung für unterschiedliche Betriebssysteme auf der Grundlage einer zentralen Datenbasis schafft [...] die Voraussetzung für die Realisierung von Single Sign-On, der einmaligen, netzweiten Anmeldung von Benutzern unterschiedlicher Dienste. Entwickler binden deshalb in zunehmendem Maße das Lightweight Directory Access Protocol (LDAP) in ihre Anwendungen ein.“ [ldp03] S. VII

Im weiteren Verlauf des Buches wird angemerkt, dass ein Verzeichnisdienst „bei der Implementierung einer zentralen Nutzer- und Diensteverwaltung unersetzlich [ist]. Das Konzept des Single Sign-On z.B. ist ohne einen zentralen Verzeichnisdienst kaum zu realisieren.“ [ldp03] S. 75

LDAP wurde 1993 als leichtgewichtige Version des bereits vorhandenen Directory Access Protocol (DAP) entwickelt. DAP ist ein Subprotokoll des X.500-Standards, der unter anderem Definitionen zur Verwaltung von Verzeichnisdaten, Verschlüsselung und Authentifizierung beinhaltet. Von X.500 gibt es verschiedene, jedoch keine vollständigen Implementierungen. Es ist gleichzeitig ein Standard der International Telecommunication Union (ITU) und der International Standardization Organization (ISO). Da X.500 auf dem komplexen 7-schichtigen OSI-Protokollstack basiert, wurde LDAP als Protokoll entworfen, das direkt auf dem einfacheren und verbreiteten TCP/IP-Stack aufsetzt.

Eine freie Implementation ist OpenLDAP von der eigens dafür gegründeten und gleichnamigen Foundation. Weitere frei erhältliche LDAP-konforme Directory Server sind OpenDS von Sun und ApacheDS. LDAP wird von einer Vielzahl von Standardprodukten unterstützt.

Active Directory Service

Seit Windows 2000 ist der Active Directory Service (ADS) als Verzeichnisdienst in Windows Betriebssystemen integriert. Active Directory basiert auf der Authentifi-

zierung über Kerberos, DNS[6] zur Speicherung von Adressen und der Benutzerverwaltung mittels LDAP. In einer hierarchischen Struktur werden Objekte wie Benutzer oder Rechner in Domänen zu logischen Gruppen zusammengefasst.

Bei der Anmeldung an einem Windows-System in einer Domäne werden die Nutzerdaten auf ein Verzeichnis, das Active Directory, abgebildet. Hier liegt ein Ansatzpunkt für die einmalige Anmeldung.

2.1.5 Sicherheitsaspekte

Im Zusammenhang mit der einmaligen Anmeldung steht die Sicherheit sehr im Vordergrund. Das Bundesdatenschutzgesetz verpflichtet zum Einsatz von Sicherheitsmechanismen, um sensible Daten zu schützen:

> *„Öffentliche und nicht-öffentliche Stellen, die selbst oder im Auftrag personenbezogene Daten erheben, verarbeiten oder nutzen, haben die technischen und organisatorischen Maßnahmen zu treffen, die erforderlich sind, um die Ausführung der Vorschriften dieses Gesetzes [...] zu gewährleisten." [bds]*

Als Vorteil von Single Sign-On wird genannt, dass das Passwort nur einmal übertragen werden muss, bei der Primärauthentisierung. Danach werden Credentials verwendet, mit denen die Echtheit der Benutzeridentität nachgewiesen wird. Doch selbst bei der einmaligen Eingabe des Passwortes besteht natürlich die Gefahr des Missbrauchs. Deswegen müssen im Firmennetzwerk geeignete Sicherheitsmaßnahmen getroffen werden.

Nach den Richtlinien aus dem Buch „Kryptographie"[7] sollte man für ein gutes, das heißt schwer zu erratendes Passwort

- **wenigstens 10 Zeichen,**
- **ein Zeichen höchstens 2 mal,**

6 Domain Name Server: löst u.a. in Netzwerken Domainnamen in IP-Adressen auf

7 vgl. [kry01b] S. 96

- **keine echten Wörter**

sowie

- **keine persönlichen Informationen** wie den Namen des Partners, Kindes, Elternteiles oder Freundes, die Telefon- oder Sozialversicherungsnummer, das Kfz-Kennzeichen oder Geburtsdatum

verwenden,

- **Groß- und Kleinbuchstaben, Zahlen, Leerzeichen, Satzzeichen und andere Symbole mischen**

und

- **das Passwort nicht aufschreiben, sondern es auswendig lernen.**

Neben dem gut gewählten Passwort ist eine wichtige Maßnahme die gesicherte Übertragung von vertraulichen Informationen. Dabei sind die drei kryptographischen Grundsätze[8] interessant:

- **Vertraulichkeit:**
  Ein Lauscher kann aus den abgehörten Daten nicht den Inhalt ermitteln

- **Integrität:**
  Die übertragenen Daten können nicht verfälscht werden
  oder Verfälschungen werden erkannt

- **Authentizität:**
  Die Daten stammen tatsächlich vom Sender,
  nicht authentische Datenpakete werden erkannt

Eine Möglichkeit, diese Kriterien umzusetzen, bietet **SSL** (Secure Socket Layer). SSL ist ein sitzungsbasiertes Protokoll zur Authentisierung und verschlüsselten Datenübertragung mit Hilfe von digitalen Zertifikaten. Es stellt auf Basis des TCP/IP-

8 vgl. [srv02] S. 152

Protokollstacks einen sicheren Kanal zwischen Client und Server her. Da es auf der Transportschicht arbeitet, ist es vom verwendeten Anwendungsprotokoll unabhängig. Die Endpunkte müssen je eine entsprechende Variante des Protokolls implementieren. Die Kommunikation geschieht über dedizierte Ports. Eine SSL-Sitzung kann dabei mehrere sichere Verbindungen umfassen. SSL wird von allen wichtigen Webbrowsern sowie einer breiten Palette von Soft- und Hardware verwendet und unterstützt. [kry01b] S. 269*ff*

TLS (Transport Layer Security) ist eine Weiterentwicklung von SSL, indem verbesserte Algorithmen, zusätzliche Protokollschichten sowie der Standardport des Servers genutzt werden.

2.1.6 Werkzeuge

Einige Technologien, die für das Verständnis der darauf folgenden Kapitel nützlich oder sogar wesentlich sind, sollen hier kurz vorgestellt werden.

Java

Java EE oder JEE ist die Abkürzung für die *Java Platform Enterprise Edition*. Es handelt sich dabei um eine Spezifikation von Sun Microsystems. Java ist eine objektorientierte Programmiersprache, die mit Hilfe der Java Runtime (JRE) plattformunabhängig ausgeführt werden kann. Im Gegensatz zur desktoporientierten Java Standard Edition (Java SE) laufen JEE-Anwendungen auf einem Server.

Die notwendigen Schnittstellen für einen zentralen Verzeichnisdienst bieten der **Java Authentication and Authorization Service** (JAAS) und das Java Naming and Directory Interface (JNDI). JAAS und JNDI sind seit Version 1.4 Bestandteil der Java Standard Edition. [jkr]

JAAS stellt eine standardisierte Schnittstelle zur Benutzeranmeldung und Berechtigung bereit. Laut [jaa] bietet JAAS unter anderem die „Möglichkeit eines Single-Sign-On an der Anwendung, in dem beispielsweise die Kennung des Betriebsystem-

Benutzers, unter dem die Anwendung ausgeführt wird, ohne weitere Benutzerinteraktion an Java »durchgereicht« werden kann.“

> *„Eine wichtige Rolle bei der Realisierung von ticketbasierten Single-Sign-On Lösungen spielt z.B. auch der Java Authentication and Authorization Service (JAAS). Bei der Verwendung von JAAS in Verbindung mit dem Kerberos Login Module kann man die benötigten Single-Sign-On Informationen im zentralen Active Directory ablegen. Mit Hilfe des Lightweight Directory Access Protocol (LDAP) ist es dann möglich, die Rollen des jeweiligen Benutzers auszulesen und hierüber die Ausführungsberechtigungen zu erteilen.“ [pea] S. 3*

JNDI dient dazu, auf beliebige Namens- und Verzeichnisdienste zuzugreifen. Für den Zugriff auf einen Dienst mit der JNDI-Schnittstelle werden neben der eigentlichen JNDI-API (Application Programming Interface) entsprechende Service-Implementierungen benötigt, beispielsweise für den Zugriff auf LDAP. [jnd]

Enterprise JavaBeans (EJB) ist ein Framework zur Erleichterung der Programmierung in verteilten, transaktionsorientierten Umgebungen. Zur Konfiguration von Anwendungen, zur Verringerung der Komplexität beim Programmieren und um festzulegen, wie Transaktionen behandelt werden sollen, werden Metadaten verwendet. Sicherheit und Transaktionen definiert man in einer Konfigurationsdatei, dem Deployment Descriptor.

EJB ist ein Standard im Bereich der Java Application Server. Das Framework bietet ein Komponentenmodell, mit welchem man für Geschäftsanwendungen typische Komponenten erstellen kann, indem man geeignete Interfaces implementiert.[9]

Aspektorientierte Programmierung (AOP) ist eine Technik, um Bestandteile einer Anwendung getrennt zu entwickeln und Codewiederholung zu vermeiden, indem der Programmcode ausgelagert wird. Ein Aspekt liegt an einer zentralen Stelle und kann mehrfach verwendet werden. Eine Möglichkeit zur Umsetzung von AOP sind Interceptoren. Diese fangen Methodenaufrufe ab und führen zusätzlichen

9 vgl. [prg03] S. 138 bzw. [spr] S. 178

Code aus. Den Aufruf der Interceptoren steuert das zugrundeliegende Framework, etwa mit Hilfe von Konfigurationsdateien.

Das **Spring**-Framework soll die Entwicklung von JEE-Anwendungen erleichtern. Durch den Einsatz von Spring können Komponenten komfortabel konfiguriert werden. Das Framework unterstützt den Aufbau von Interceptorketten [lmk]. Eine Ergänzung zu Spring für Authentisierung und Autorisierung bietet Spring Security, wodurch Sicherheitsaspekte für Geschäftsanwendungen implementiert werden können.

Web Services

Der BLS bietet unter anderem Web Service-Schnittstellen für die enthaltenen Komponenten an. **Web Services** (WS) sind Dienste, bei denen die Kommunikation zwischen Client Anwendungen und Servern über standardisierte Protokolle (HTTP, SOAP) und Dateiformate (XML) stattfindet. Sie basieren auf den Komponenten UDDI (Universal Description, Discovery and Integration) als WS-Verzeichnis, WSDL (Web Services Description Language) als Schnittstelle und einem Protokoll wie SOAP (Simple Object Access Protocol) oder XML-RPC (Remote Procedure Call)[10] zur Übermittlung von Prozeduraufrufen. Web Services sind plattform- und programmiersprachenunabhängig.

SAML (Security Assertion Markup Language) ist eine XML-basierte Beschreibungssprache, gehört zu den Web Services und dient dem sicheren Austausch von Authentifizierungs- und Autorisierungsinformationen zwischen den Sicherheitssystemen und E-Business-Plattformen von Partnern. [it]

Software Tools

Auf einem **Application Server** laufen Programme und Dienste, deren Funktionalität über das Netzwerk zur Verfügung gestellt wird. Sie sind ihrerseits innerhalb eines

10 dt. entfernter Prozeduraufruf; Daten werden in XML-RPC per HTTP transportiert

Server-Betriebssystems installiert. Der Leasman Business Logic Server ist als Komponente in einem JEE konformen Application Server konzipiert. Beispiele für Vertreter nach dem Java Enterprise Edition Standard sind Sun GlassFish, JBoss Application Server oder IBM WebSphere. Einige Single Sign-On Systeme sind für die Installation in einem Application Server vorgesehen. Für manche Systeme ist, vor allem in der Entwicklungsphase, ein Servlet-Container wie Apache Tomcat ausreichend.

Die **Eclipse** IDE (Integrated Development Environment) stellt ein mächtiges Werkzeug zur Softwareentwicklung dar. Ausgehend von der Programmierung mit Java bietet sie Unterstützung für viele weitere Sprachen und Software-Technologien.

2.2 Beschreibung der Geschäftsanwendung

2.2.1 Der Leasman Client

Der Leasman Client ist eine Leasing-Software, die bei der DELTA proveris AG seit 1997 im Sybase PowerBuilder entwickelt wird und auf Windows-Systemen läuft. Die Entwicklung mit PowerBuilder geschieht mit einer visuellen Oberfläche und der produkteigenen Skriptsprache PowerScript.

Beim Start der Anwendung erfolgt nach der Eingabe von Nutzername und Passwort die Auswahl eines Mandanten. Welcher Mandant für den jeweiligen Nutzer verfügbar ist, steht in einer Oracle bzw. IBM Informix Datenbank. Abhängig vom gewählten Mandanten kann eine Vielzahl von Vorgängen im Leasinggeschäft von der Angebotserstellung über die Service-Abwicklung bis hin zum Verkauf bearbeitet werden.

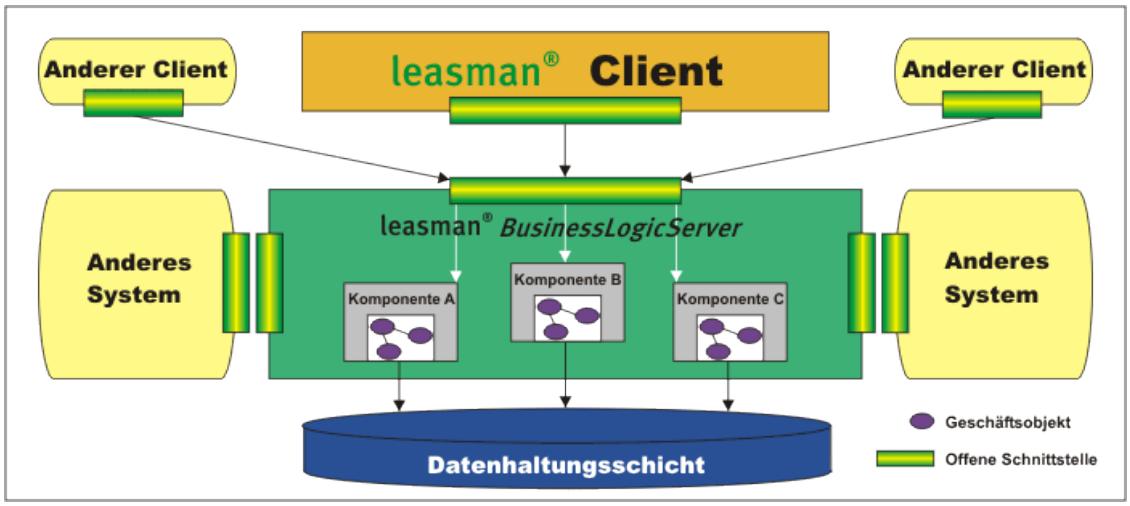


Abb. 2-1: Leasman Mehrschichtarchitektur [dpb]

2.2.2 Der Business Logic Server

Der Leasman Business Logic Server (BLS) stellt die Mittelschicht zwischen Datenbank und Präsentationsschicht dar. Er ist als Komponente im JBoss oder anderen Application Servern konzipiert. Der BLS bietet Webservice-Schnittstellen für Leasman sowie andere Clients und Systeme und hält verschiedene Komponenten mit

Geschäftsobjekten. Durch die Einführung des BLS als Mittelschicht werden die Integrationsmöglichkeiten beim Anwender erweitert.

> *„Die Serverseite wird mit Java implementiert. Als Basis für den entfernten Aufruf wird eine Enterprise JavaBean (EJB) verwendet. Dies bietet neben der Möglichkeit verschiedener Protokolle (z.B.: RMI, IIOP) und der direkten Kommunikation mit Objekten, auch die Bereitstellung einer Webserviceschnittstelle. Die eigentliche Funktionalität wird mit dem Komponentenframework Spring implementiert, in das die EJB delegiert.“ [lmk]*

Die Anbindung des BLS erfolgt über eine HTTP/SOAP-basierte Webservice-XML-Schnittstelle. Lauffähig ist der BLS in jedem Java Enterprise Edition-konformen Application Server, z.B. JBoss Application Server, IBM WebSphere, BEA WebLogic und auf allen Java 1.5-kompatiblen Hardware-Plattformen. [dpb]

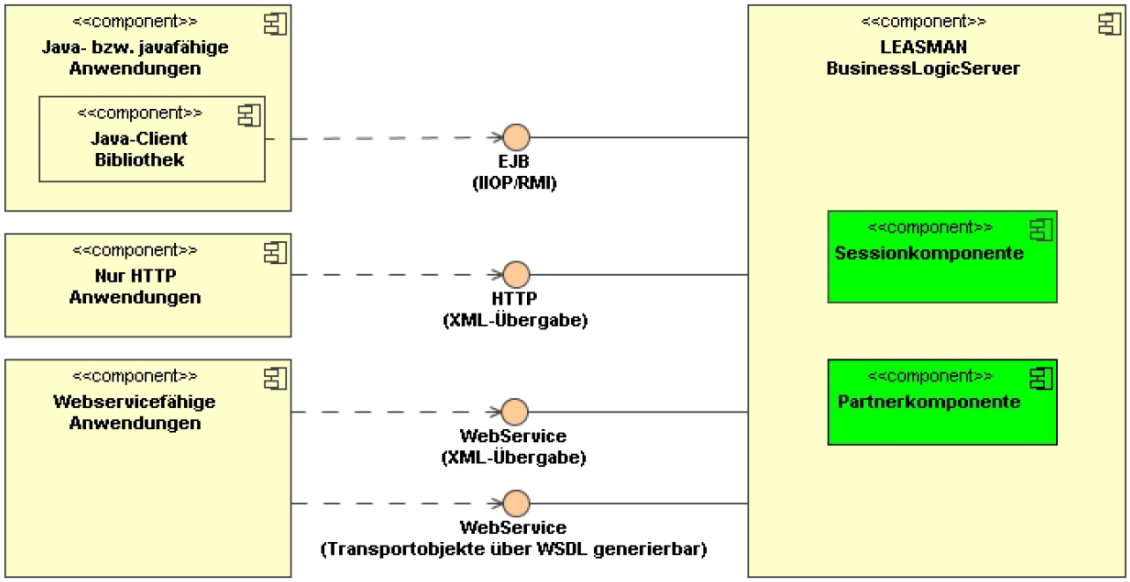


Abb. 2-2: BLS Schnittstellentypen, © DELTA proveris AG 2008 [bls]

Seit Leasman Version 6.01, dessen Release im September 2009 erfolgte, ist der Business Logic Server obligatorischer Bestandteil der Leasman-Auslieferung. Die Geschäftslogik in Form von Komponenten wird im Laufe der nächsten Releasezyklen sukzessive vom Client in den Server ausgelagert.

2.2.3 Status der Anmeldung

Folgendes Szenario beschreibt den aktuellen Anmeldevorgang bei Verwendung des Leasman mit BLS:

Ein Mitarbeiter M der Firma F, die Leasman einsetzt, meldet sich an seinem Arbeitsplatz-PC an. Dabei muss er sich natürlich zunächst an der Loginmaske des Betriebssystems mit Nutzername und Passwort authentifizieren. Nach erfolgreicher Anmeldung startet er den Leasman Client, wobei ein weiterer Anmeldevorgang erfolgt, da es sich um eine Mandanten-basierte Software handelt. Die Firma F verwendet zwei weitere Anwendungen, die auf den Business Logic Server zugreifen und dessen Geschäftslogik nutzen. Mitarbeiter M startet diese und muss noch zwei mal seine Logindaten eingeben.

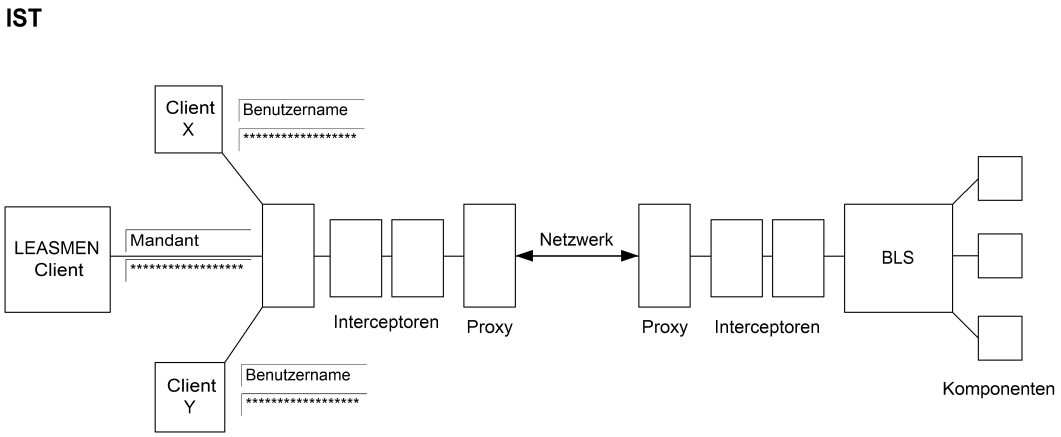


Abb. 2-3: Status BLS-Anmeldung

Aus technischer Sicht erfolgt die Anmeldung über eine Kette von Interceptoren. Bei einer Anfrage des Client an den BLS wird clientseitig im SessionInterceptor überprüft, ob bereits eine Session-ID (SID) für die Anmeldung am Server verfügbar ist. Im Erfolgsfall wird die SID dem Kommandotransportobjekt, welches den Aufruf der Geschäftslogik kapselt, übergeben und an den BLS geschickt. Ist beim Client keine SID abrufbar, wird der Request ausgesetzt und die Anmeldung eingeschoben. Es wird über die Methode `createSession()` eine Session-ID erzeugt und dann wie bereits beschrieben weiterverfahren. Die Anfrage durchläuft als Transportobjekt die Request-Kette und wird auf der Serverseite wiederum auf das Vorhandensein der

Session-ID geprüft. Mit gültiger SID wird schließlich die Anfrage des Clients bearbeitet und das Ergebnis zurückgesendet.

In eine Single Sign-On Umgebung integriert könnten die Anmeldevorgänge bis auf die Primärauthentisierung reduziert werden, indem ein Ticket erstellt wird, welches von den Clients zur Authentisierung gegen den BLS verwendet werden kann.

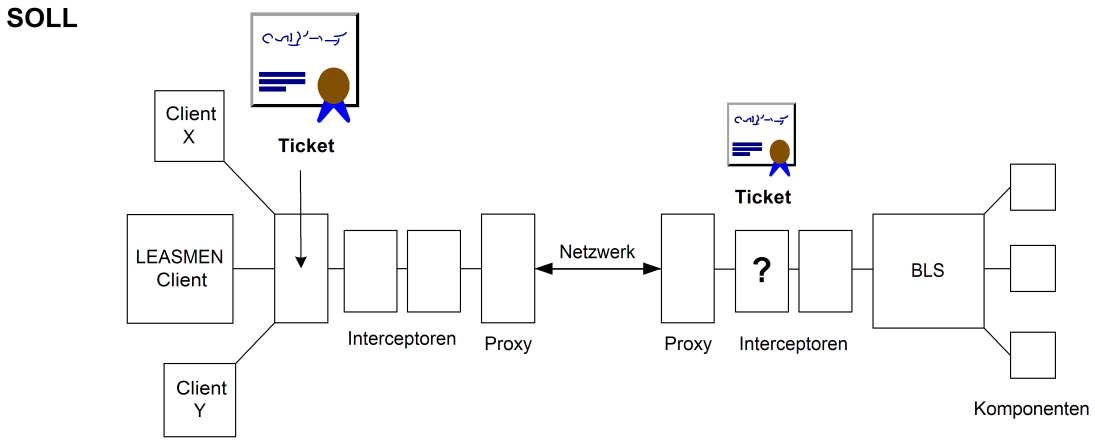


Abb. 2-4: BLS-Anmeldung per Ticket

3 SSO-Lösungen

Der erste Abschnitt dieses Kapitels stellt Strukturen vor, welche einzeln oder in Kombination eine Basis für Single Sign-On Systeme bilden. Konkrete Lösungen in Form von auslieferbaren Softwareprodukten werden anschließend thematisiert.

3.1 Ansätze

SSO-Lösungen werden je nach verwendeter Literaturquelle unterschiedlich kategorisiert. Im Folgenden werden zwei Einteilungen vorgenommen und im Anschluss konkrete Ansätze sowie eine exemplarische SSO-Session betrachtet.

3.1.1 Zentral oder Verbund

Mit einem **SSO-Server** ist ein zentraler Rechner im Netzwerk mit spezieller Funktionalität gemeint, über den die Authentisierung erfolgt. Zur Validierung der Nutzerdaten besteht eine Datenbankverbindung oder per LDAP der Zugriff auf einen

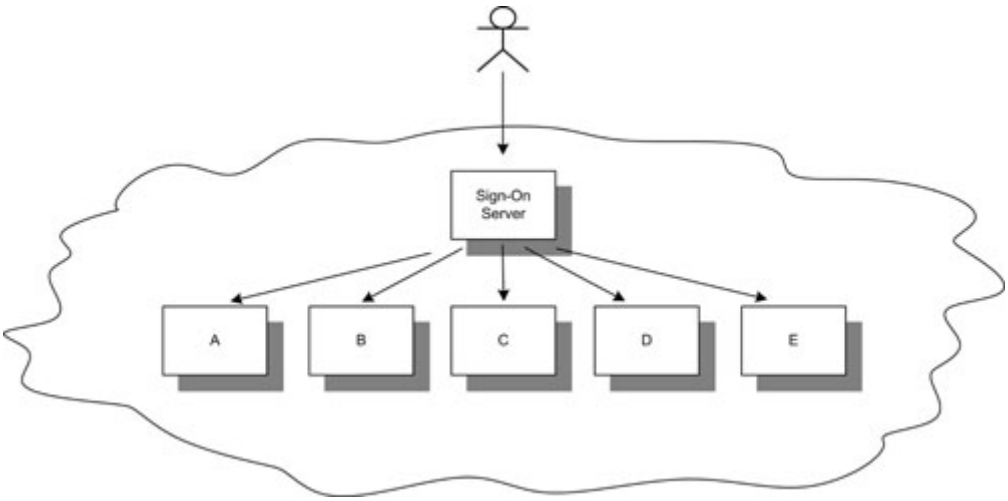


Abb. 3-1: zentraler SSO-Server [sso]

Verzeichnisdienst. Dieser Server kann dem authentisierten Nutzer ein Ticket ausstellen oder direkt zu einem Application Server weiterleiten.

Der **Circle of Trust** ist ein Verbund aus Anwendungen, die sich gegenseitig vertrauen. Mit der Anmeldung bei einer dieser Anwendungen wird der Benutzer automatisch für den kompletten *Circle* angemeldet. Die Vertrauensbasis kann dabei ein Ticket sein, das bei der ersten Anmeldung erstellt wird und mit dem der Nutzer innerhalb des Verbundes authentifiziert werden kann. Im Gegensatz zum zentralen SSO-Server handelt es sich hierbei um einen dezentralen Ansatz.

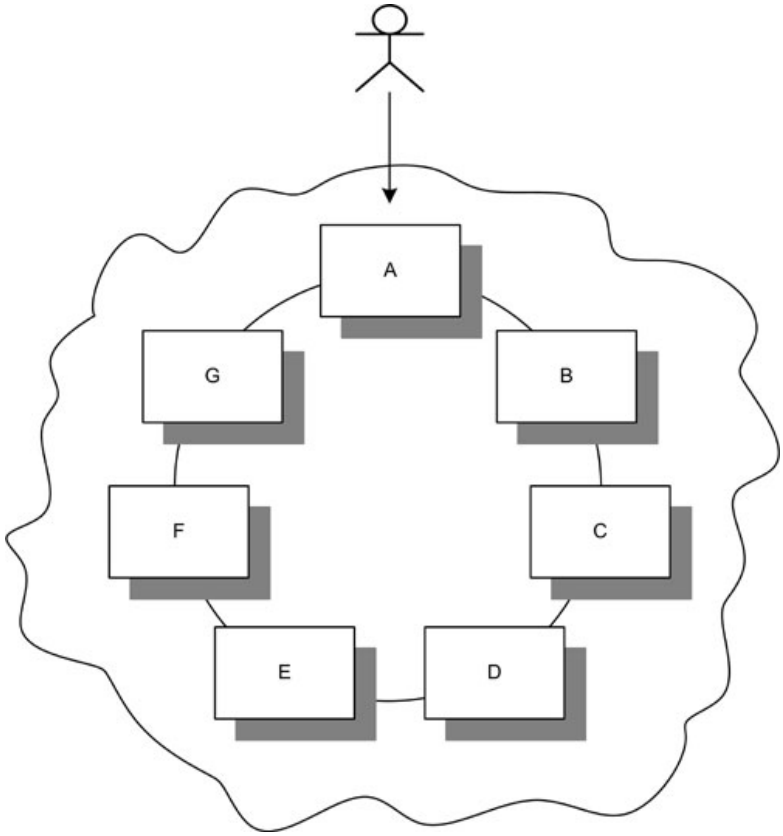


Abb. 3-2: Circle of Trust [sso]

3.1.2 Lokal, Portal oder E-SSO

Bei den **lokalen** oder **clientbasierten** Lösungen werden die Nutzerdaten lokal auf dem Rechner gehalten. Meist wird eine spezielle Software installiert. Diese automatisiert die Anmeldevorgänge am Arbeitsplatz. Solche Lösungen werden auch Appli-

ances genannt. Beispiele dafür sind Passwort-Manager, wie sie etwa in Webbrowsern eingesetzt werden oder das Microsoft Identity Metasystem.

Die **Portal**lösungen, auch als **Web-SSO** bezeichnet, verwenden HTTP-Cookies. Im Gegensatz zum zentralen Enterprise-SSO wird hier eher der Ansatz des Circle of Trust verfolgt. Authentisierung und Autorisierung laufen hier getrennt ab. In diesem Zusammenhang steht auch der Begriff „Identity Federation". Bei [it] ist darüber zu lesen:

> *„Die Identity Federation ermöglicht es Unternehmen und Organisationen, vertrauenswürdige Identitäten anderer Organisationen, wie zum Beispiel von Partnern oder Zulieferern, zu akzeptieren. Das Ziel von Federation ist es, Informationen von Identitäten über Unternehmensgrenzen hinweg zu integrieren, um Geschäftsprozesse zu vereinfachen."*

Web Services und SAML werden verwendet. MSN Passport bzw. Windows Live ID sind hier als Beispiele zu nennen.

SSO-Lösungen, die **serverbasiert** sind, werden auch als **Enterprise-SSO** (E-SSO) bezeichnet. Dabei dienen zentrale Credentials (Tickets) zur Authentisierung, wobei als SSO-Server etwa ein Kerberos Key Distribution Center verwendet werden kann. Als Beispiel eines SSO-Systems ist hier CAS (Central Authentication Service) zu nennen, welches im Kapitel 3.2.1 noch näher betrachtet wird.

In der vorliegenden Arbeit wird der Schwerpunkt auf den letztgenannten Ansatz, die Enterprise-SSO-Systeme gelegt, da es sich bei Leasman um eine unternehmensinterne Geschäftsanwendung handelt.

3.1.3 Kerberos

Eine Methode, die den Ansatz des zentralen SSO-Servers verfolgt, ist Kerberos. Der Name entstammt der griechischen Mythologie. Danach ist Kerberos ein dreiköpfiger Hund, der den Eingang zum Hades bewacht. Die Funktion des SSO-Servers – im Kerberos-Kontext wird auch vom Authentifizierungs-Server (AS) gesprochen – übernimmt hier das sogenannte *Key Distribution Center* (KDC).

Bei der Primärauthentisierung gibt der Nutzer seine Logindaten ein, woraufhin ein Request an den KDC erfolgt. Der Server sendet ein verschlüsseltes Service Ticket mit einem Sitzungsschlüssel (Session Key) zurück, mit dem der Client sich nun bei Anwendungen, welche die Kerberos-Technologie unterstützen, anmelden kann.

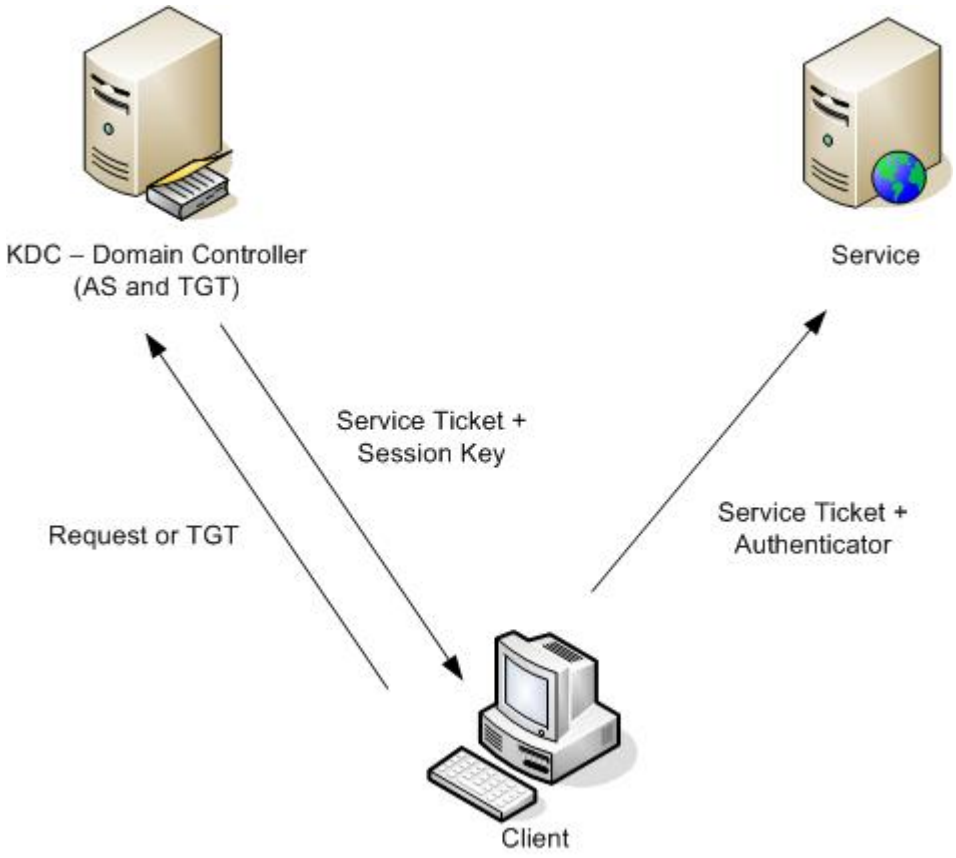


Abb. 3-3: Kerberos [ado]

Kerberos ist kompatibel zu verschiedenen Betriebssystemen. Entwickelt am Massachusetts Institute of Technology (MIT) und seit vielen Jahren im UNIX-Bereich eingesetzt, wird die aktuelle Kerberos Version 5 auch von Windows-Systemen unterstützt. So kann von Windows-Rechnern auf UNIX-Ressourcen und umgekehrt zugegriffen werden, ohne dass eine weitere Authentifizierung nötig ist. [win01]

Ein Punkt, der im Zusammenhang mit Kerberos beachtet werden muss und gegenüber anderen Lösungen eventuell als Nachteil gesehen werden kann, ist, dass jede

Anwendung *kerberisiert* sein muss, die innerhalb des SSO-Systems verwendet werden soll:

> *„Damit eine Anwendung Kerberos verwenden kann, muss sie programmtechnisch so angepasst sein, dass die geeigneten Aufrufe an die Kerberos Bibliotheken gesendet werden können. Hierbei stellen vor allem die so genannten Closed Source Anwendungen ohne standardmäßigen Kerberos-Support den problematischen Teil dar." [pea] S. 3*

Da ein zentraler Server zur Authentifizierung in unternehmensübergreifenden Netzwerken als einziger Angriffspunkt ein Sicherheitsrisiko darstellt, ist der Kerberos-Einsatz vorrangig für den Gebrauch innerhalb von Unternehmen bestimmt.

3.1.4 X.509

Ein weiterer Ansatz ist die Verwendung von **digitalen Zertifikaten** nach dem X.509-Standard. Auch hier gilt aber, dass Anwendungen diese unterstützen müssen. Außerdem müssen die Zertifikate jeweils noch auf Benutzerkonten in Verzeichnissen abgebildet werden. Der administrative Aufwand ist hier, wie auch bei Kerberos beträchtlich. [1pw]

3.1.5 Identity Federation

Der Vorteil des Einsatzes von Identity Federation ist, dass dabei standard-basiert unter Nutzung von Web Services gearbeitet wird. Federation ist daher, beispielsweise im Gegensatz zu Kerberos, auch über die Unternehmensgrenzen hinweg einfach einsetzbar. Und Federation ist sowohl bei web-basierenden als auch anderen Anwendungen vergleichsweise einfach zu implementieren. [1pw]

3.1.6 Ticketbasiertes SSO

Das ticketbasierte SSO ist am weitesten verbreitet [kus]. Vereinfacht dargestellt läuft eine Session, in der Single Sign-On auf Ticketbasis verwendet wird, folgendermaßen ab:

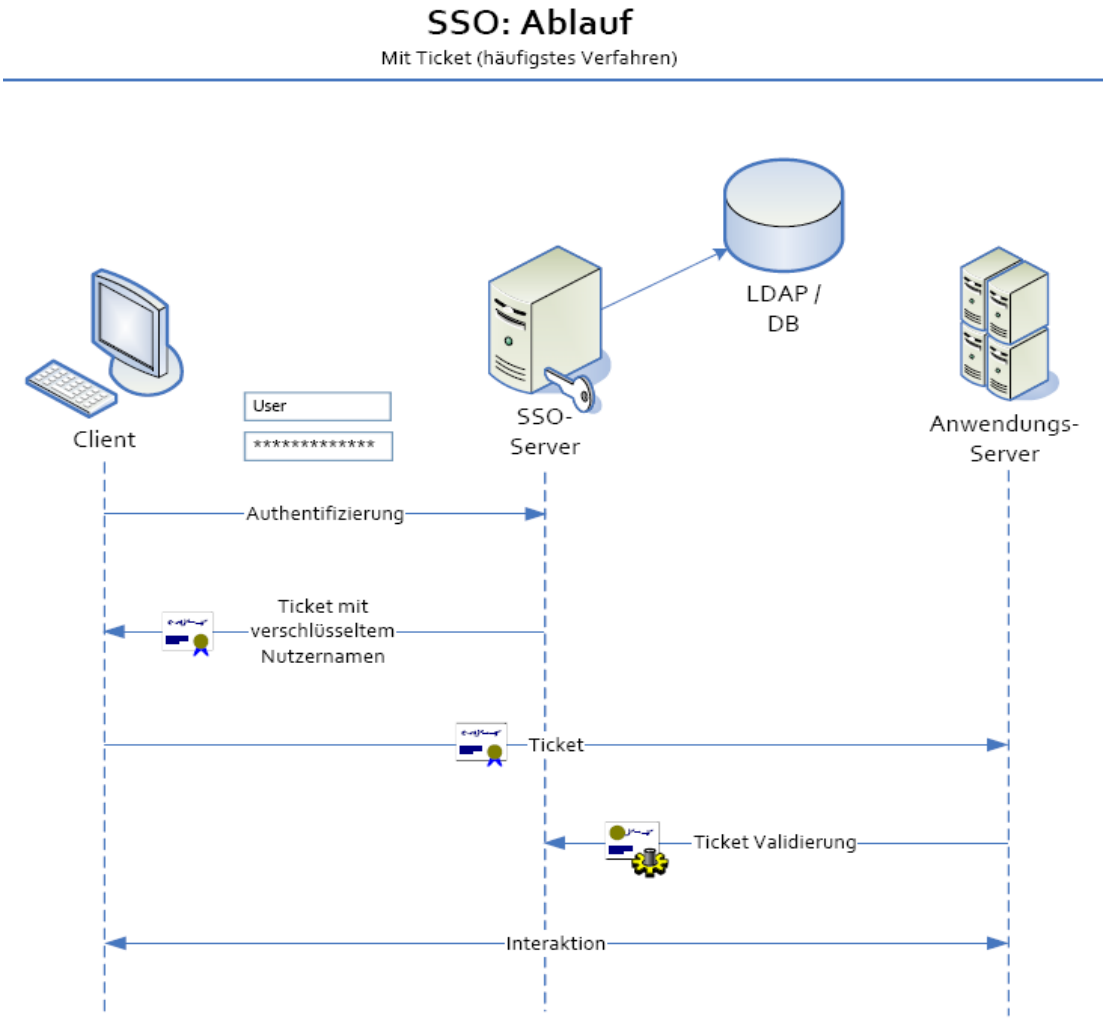


Abb. 3-4: SSO-Session mit Ticket

Eine Person authentifiziert sich im Rahmen einer Client-Anwendung mit Nutzername und Passwort an einem SSO-Server. Über eine Datenbankverbindung beziehungsweise den Zugriff auf einen Verzeichnisdienst, beispielsweise mit LDAP werden die Nutzerdaten validiert. Der Server erstellt ein Ticket mit dem verschlüsselten Benutzernamen und schickt dieses an die aufrufende Anwendung zurück.

Der Nutzer kann sich nun an allen Anwendungsservern mit dem Ticket anmelden, die dieses akzeptieren. Die entsprechenden Anwendungen oder Dienste senden das Ticket zur Validierung an den SSO-Server und bei übereinstimmenden Nutzerdaten können die angeforderten Leistungen in Anspruch genommen werden, bis der User sich vom zentralen Server abmeldet oder die Session abläuft.

3.1.7 Single Log-Out

Einige SSO-Systeme bieten eine Single Log-Out oder Single Sign-Out Funktion an. Wenn die Session entweder durch den User oder über das Ende der Laufzeit beendet wird, werden alle Anwendungen im Rahmen der SSO-Umgebung abgemeldet.

3.2 SSO-Systeme im Java Umfeld

Auf Grundlage der in 3.1 genannten Ansätze wird eine Vielzahl von Produkten angeboten. Dieser Abschnitt stellt einige solcher Single Sign-On Systeme vor. Dabei werden nur Systeme betrachtet, die auf Java basieren und quelloffen sowie frei erhältlich sind, also keiner kommerziellen Lizenz unterliegen. Darüber hinaus kommen nur Lösungen in Frage, die in verteilten Anwendungen eingesetzt werden können. Lokale Systeme fallen somit heraus, da hier die Serverkomponente fehlt. Bei diesen lokalen Lösungen könnte die Anmeldung am Leasman Client zwar automatisiert erfolgen, indem ein Passwortmanager die Logindaten einträgt. Allerdings wäre damit das Ziel nicht erreicht, den Einstiegspunkt für die Anmeldung auf den Server zu legen und damit weitere Clients integrieren zu können.

Bedingt durch die beschriebenen Vorgaben wurden die folgenden Systeme ausgewählt. Zunächst werden sie in Unterkapiteln einzeln vorgestellt und anschließend miteinander verglichen. Schließlich wird eine Auswahl aus den SSO-Systemen getroffen, für die im Weiteren ein Prototyp entworfen und implementieren werden soll.

3.2.1 JA-SIG CAS

Der Central Authentication Service (CAS) von der Java Architectures Special Interest Group (JA-SIG) basiert auf dem Spring-Framework. CAS verwendet den Ansatz des zentralen SSO-Servers, der die Authentisierungen behandelt. Es bietet Sicherheit durch die Verwendung von HTTPS und unterstützt SAML sowie Spring Security.

Die **Architektur** von CAS gleicht dem Aufbau von Kerberos. Es gibt einen CAS Server und CAS Clients. Anwendungen werden als Services bezeichnet und müssen CASified, das heißt, für die Verwendung durch CAS angepasst werden, indem ein CAS Client eingebunden wird.

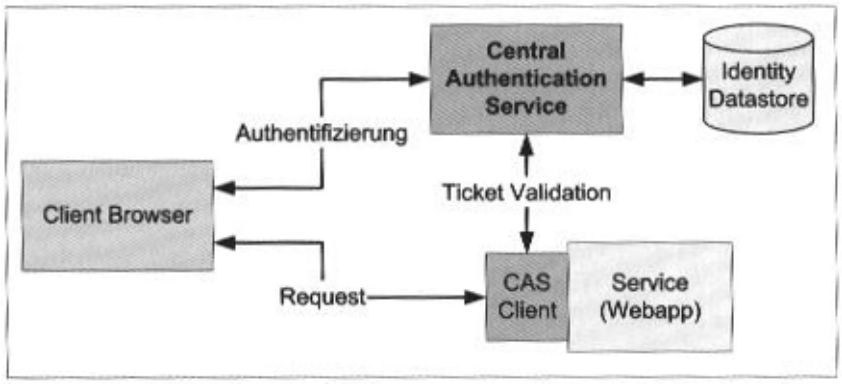


Abb. 3-5: CAS Architektur [jma] S. 22

Der Client prüft, ob der Benutzer bereits authentifiziert ist und leitet zum Server weiter, wenn das nicht der Fall ist. Der Server verlangt die Benutzerdaten und erzeugt nach erfolgreicher Authentifizierung, ähnlich wie bei Kerberos, zwei verschiedene **Tickets**. Zum einen erstellt er das Service Ticket (ST), das eine Berechtigung speziell für eine Anwendung und einen Benutzer ist. Außerdem gibt es ein Ticket Granting Cookie (TGC), welches zur Erkennung eines bereits authentifizierten Nutzers gegenüber anderen CAS Clients dient. Mit den beiden Tickets wird ein Redirect zum Service ausgeführt. Nachdem der CAS Client das ST durch den Server verifiziert hat, erhält der Nutzer Zutritt zur Anwendung, die eine eigene Sitzung aufbaut. Das Service Ticket wird durch den Server gelöscht.

CAS beschränkt sich lediglich auf die Verarbeitung von Authentifizierungsvorgängen. Die Behandlung der Autorisierung bleibt den Anwendungen überlassen, was teilweise als Manko gesehen wird[11]. Andererseits wird die SSO-Lösung dadurch bewusst schlank gehalten. Der Einsatz von CAS sollte daher nur bei Anwendungen erwägt werden, die ihrerseits ein umfassendes Autorisierungsmanagement implementieren.

11 vgl. [jma] S. 20*ff*

Als Mechanismen zur **Authentifizierung** integriert CAS unter anderem JAAS, JDBC, LDAP und X.509 Zertifikate. Durch CAS werden auch Single Log-Out und die Remember-Me Authentifizierung unterstützt.

Die **Auslieferung** des CAS Servers, der als Java Servlet implementiert ist, erfolgt zusammen mit Erweiterungen zur Systemintegration als Web Application Archive. Er kann daher in einem Servlet-Container wie Apache Tomcat deployt und ausgeführt werden. Als Client APIs werden beispielsweise Java, .NET, PHP und Ruby angeboten.

Die **Dokumentation** von CAS ist unvollständig und nicht einheitlich, wodurch ein zusätzlicher Aufwand bei der Recherche zu speziellen Problemstellungen notwendig wird.

CAS findet bevorzugt im akademischen Bereich Anwendung, begründet dadurch, dass es ursprünglich von der Yale University entwickelt wurde.

3.2.2 JOSSO

Wie CAS ist auch JOSSO (Java Open Single Sign-On) Spring-basiert. JOSSO kann auch als zentralisierte und plattformunabhängige SSO-Infrastruktur bezeichnet werden. Es unterstützt „starke Authentisierung“ durch die Nutzung von X.509 Client-Zertifikaten. Es basiert auf Standards wie JAAS, Web Services via SOAP, Enterprise Java Beans und dem Struts Framework mit Servlets und JSP.

JOSSO kann als ZIP-Archiv heruntergeladen werden. Darin enthalten ist auch eine Kommandozeile, die JOSSO Deployment Console, für die Installation von JOSSO Gateway und Agents. Außerdem wird ein Archiv zum Download angeboten, das einen Apache Tomcat mit bereits installiertem JOSSO Gateway inklusive Beispielanwendung enthält. Es muss nur noch entpackt und der Server gestartet werden. Neben Apache Tomcat werden die gängigen Application Server wie JBoss, Weblogic und Websphere unterstützt.

Der JOSSO Gateway stellt einen zentralen SSO-Server dar und wird auch als Identity Provider bezeichnet. Der Gateway konfiguriert Identity, Credential und Session Store und legt fest, welches Authentication Scheme verwendet wird. Zur Konfiguration dieser Komponenten werden in XML-Dateien Beans definiert und referenziert. Beispielsweise wird in den Einstellungen für die Bean `josso-identity-store` festgelegt, woher die Nutzerdaten bezogen werden sollen. Als Identity Store können LDAP und Datenbank Systeme eingesetzt werden.

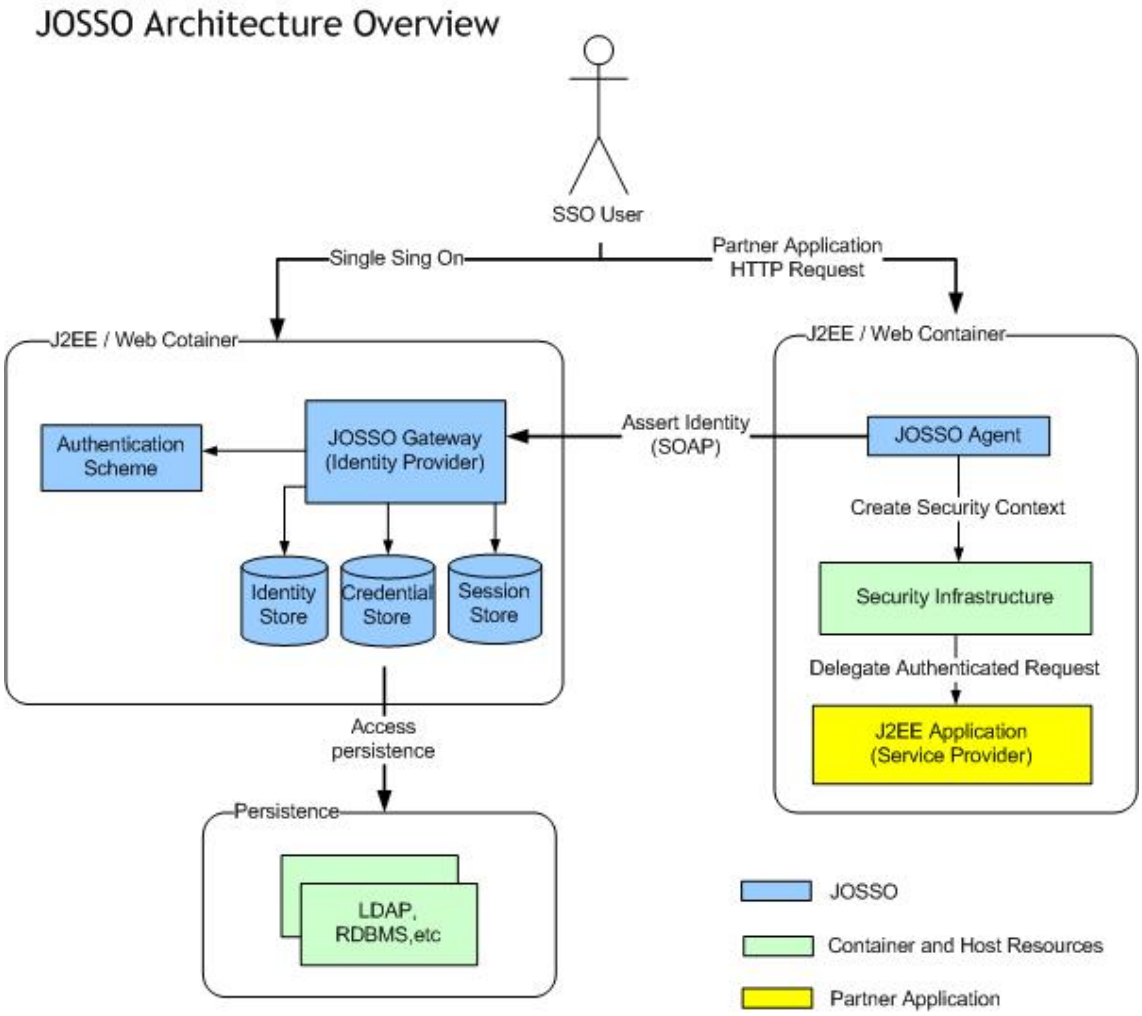


Abb. 3-6: JOSSO Architektur [jsa]

Unterstützte Authentifizierungsformen sind *basic*, *strong*, *bind* und *remember-me*. Als besondere Form wird *Windows Authentication* unterstützt. Dabei geschieht die Anmel-

dung über die Nutzerdaten des Betriebssystems. JOSSO ermöglicht fein granulierte Autorisierung durch die Unterstützung von Spring Security.

Als Agent wird bei JOSSO ein SSO-Client bezeichnet. Dieser nimmt Anfragen eines Users an die Anwendung (Partner Application) entgegen und kümmert sich um die Authentifizierung. Per SOAP kommuniziert er mit dem Gateway, um erforderliche Nutzerdaten zu erhalten und die SSO Session zu validieren. Die Anwendung, deren Dienste der Nutzer in Anspruch nehmen möchte, wird als Service Provider bezeichnet. JOSSO stellt zur Implementierung von eigenen Clients eine API bereit. Diese ist in der Auslieferung enthalten.

Zur aktuellen JOSSO Version 1.8.1 ist keine API Dokumentation verfügbar. Lediglich zur Version 1.7 von 2008 existiert eine Dokumentation[12]. Es besteht aber die Möglichkeit, das JOSSO Repository[13] zu durchsuchen und die Quelldateien anzuschauen. Dort ist immer die aktuelle Version verfügbar. Allerdings ist das Navigieren durch die Baumstruktur der API langsam und weniger übersichtlich als eine Java API Dokumentation. Für die Unterstützung der Anwender existieren zu JOSSO ein Forum sowie kommerzieller Support.

3.2.3 Sun OpenSSO

Das Open Web SSO-Projekt (OpenSSO) von Sun Microsystems unterliegt der Common Development and Distribution License (CDDL). Es basiert auf dem kommerziellen Sun Java System Access Manager. Als bevorzugter Application Server wird GlassFish, ebenfalls ein Open-Source-Projekt von Sun, angegeben.

OpenSSO hat das Ziel, mit Hilfe von Core Identity Services die Verwendung von Single Sign-On als Sicherheitskomponente in einer Netzwerkstruktur zu erleichtern. Es schafft eine Grundlage für die Einbindung von Webanwendungen, die mit Daten aus heterogenen Identitätsspeichern arbeiten und auf Web- oder Application Ser-

12 siehe http://www.java2s.com/Open-Source/Java-Document/Authentication-Authorization/josso-1.7/Catalogjosso-1.7.htm

13 siehe http://fisheye.josso.org/browse/JOSSO/

vern laufen [osd]. Es vereint Identitäts- und Passwortmanagement, Zugriffskontrolle, Single Sign-On und Federation.

OpenSSO wird in verschiedenen **Varianten** zur Verfügung gestellt. Die Enterprise-Variante ist ausführlich getestet und wird etwa jährlich in einer neuen Version veröffentlicht. Dagegen wird die Express-Version weniger intensiven Tests unterzogen, enthält neue Features und die Veröffentlichung geschieht in kürzeren Abständen. Express und Enterprise haben die selbe Codebasis. Für beide Varianten wird ein kommerzieller Support durch Sun Microsystems angeboten.

Das von der Projektseite herunterzuladende Archiv enthält unter anderem den SSO Server als WAR zum Deploying in einem Application Server. In die **Auslieferung** integriert ist auch ein eigener Nutzer- und Konfigurationsdatenspeicher: OpenDS von Sun. Neben diesem können auch andere LDAP-konforme Data Stores verwendet werden. Das Deployment beinhaltet außerdem das OpenSSO Client SDK als JAR zur Entwicklung von Client Anwendungen.

Clients können Web und Non-Web Applikationen sein, also im Browser oder als eigenständige Anwendung auf dem Desktop laufen. [tec] S. 23

OpenSSO beinhaltet Authentication, Authorization, Session, SAML, Federation und Logging Services:

Der **Authentication** Service bietet die Funktionalität, Nutzerdaten abzufragen und sie gegen einen bestimmten Authentifizierungsdatenspeicher zu validieren. Im Falle einer erfolgreichen Authentifizierung erstellt der Service ein Session Token für den User, das bei allen Anwendungen innerhalb der SSO Umgebung gültig ist. Eine Vielzahl von Authentifizierungsmodulen wird durch OpenSSO Enterprise unterstützt, zudem können weitere Module unter Verwendung des JAAS Service Provider Interface (SPI) eingebunden werden [tec] S. 30. Mit Hilfe des integrierten JAAS-Frameworks können Authentifizierungsmodule auch verkettet werden [jbo]. Durch die-

se Kombination von mehreren Authentifizierungsverfahren lassen sich Nachteile einzelner Module ausgleichen und die Sicherheit erhöhen [ssv].

Der **Authorization** oder **Policy Service** wertet die Nutzerrechte aus und entscheidet anhand von einstellbaren Regeln über den Zugriff auf geschützte Ressourcen. [jbo]

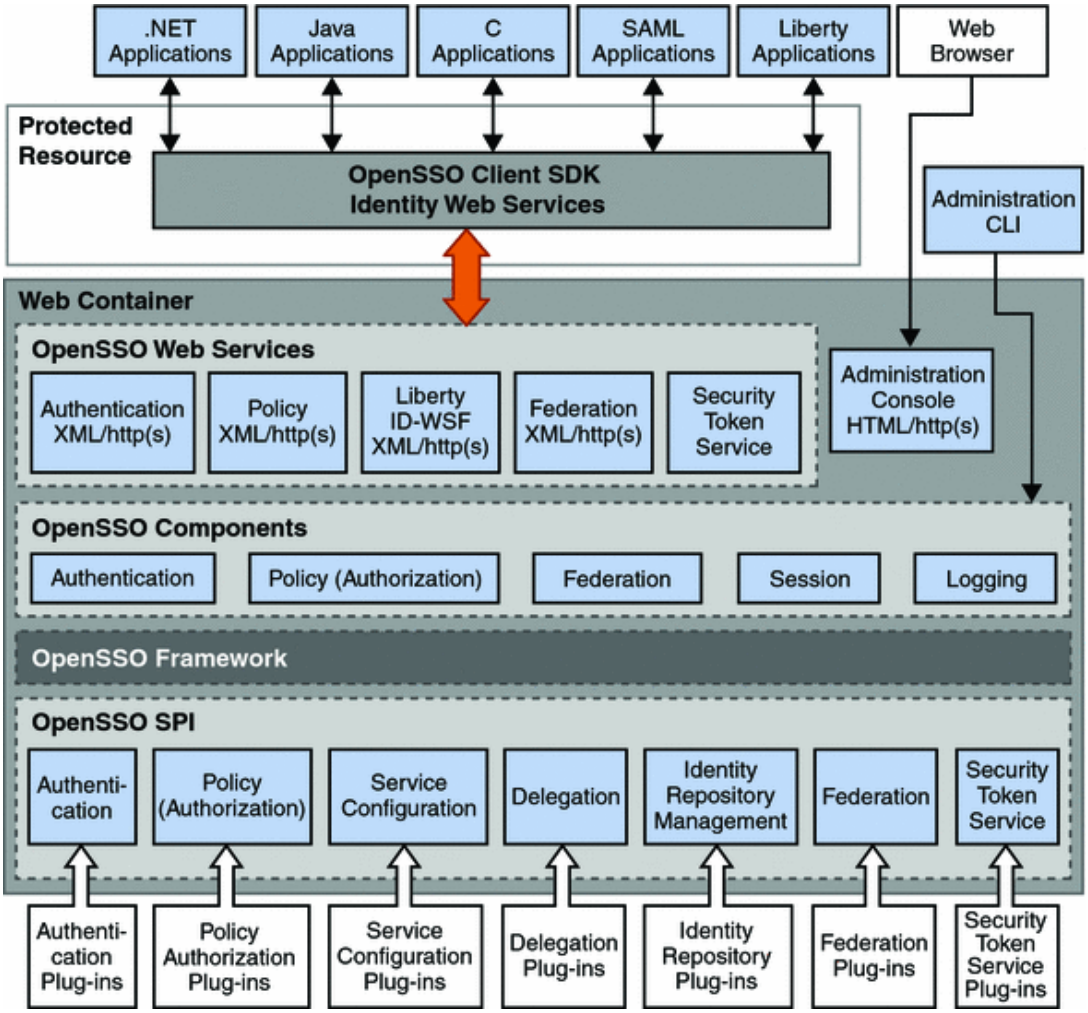


Abb. 3-7: OpenSSO Architektur, [tec] S.26

Der **Session** oder **SSO Service** sichert dauerhaft die Nutzeridentität und ermöglicht damit den Zugriff auf mehrere Ressourcen innerhalb der selben Domain, ohne dass sich der Nutzer jedes Mal interaktiv authentifizieren muss. Außerdem stellt der Service *cross-domain single sign-on* (CDSSO) zur Verfügung um den Zugriff auf An-

wendungen Domain-übergreifend zu realisieren [jbo]. Das Session Token, auch als Session-ID bezeichnet und in der API als *SSOToken* benannt, ist ein verschlüsselter einmaliger String, der die Session identifiziert. Das Token wird beim Zugriff auf geschützte Ressourcen zum Abfragen der Nutzerinformationen verwendet [tec] S. 80. Der Access Manager von OpenSSO speichert die Sessions in Hashtables im Speicher der Java Virtual Machine (JVM). Er läuft auf dem Server, bei welchem sich der Nutzer zuerst eingeloggt hat. [sam]

Zur Gewährleistung der Sicherheit ist es wichtig, dass die gesamte SSO Session beendet wird, wenn ein Nutzer die Anwendungen schließt, sodass sie nicht durch Dritte missbraucht werden kann. Hierfür integriert OpenSSO Single Sign-Out.

Der **SAML Service** erlaubt Geschäftspartnern den sicheren Austausch von Informationen zur Authentisierung und Autorisierung über das Internet.

Der **Identity Federation Service** führt eine Vielzahl von lokalen Identitäten in einer Federated Identity zusammen. Das gibt dem Nutzer die Möglichkeit, sich ohne Re-Authentisierung zwischen verschiedenen Service Providern zu bewegen. Dieses Feature baut auf dem Konzept des Circle of Trust auf.

Der **Logging Service** stellt eine Kontrollfunktion zur Verfügung, indem er Informationen aufzeichnet, wie etwa bei Zugriffsverweigerungen, Authentisierungsvorgängen und Verletzung von Nutzerrechten. [jbo]

OpenSSO stellt eine **WebService**-Schnittstelle bereit und bietet Sicherheit, indem die Verbindung zwischen Web Service Client und Provider durch Security Agents überwacht wird. [ssv]

Die SSO-Lösung implementiert **Sicherheitsaspekte** wie Account Locking, um nach einer Anzahl von Fehleingaben den Account zu sperren. Die Konfiguration des SSO-Servers ist nur mit Administratorrechten möglich, sodass unautorisierte Änderungen unterbunden werden. Vom System verwendete Nutzerdaten werden verschlüsselt. [tec] S. 105

Die **Dokumentation** zum SSO System ist sehr umfangreich. Es steht eine Reihe von Dokumenten als Website und PDF zur Verfügung, beispielsweise ein *Technical Overview*, *Administration Guide* oder *Developer's Guide*. Sehr hilfreich für die Entwicklung ist auch die *Java API Reference*. Durch die Vielzahl von Dokumentationen entsteht allerdings das Problem, Antworten auf spezielle Fragen zu finden. Die vorhandenen Tutorials basieren auf zusätzlichen Dokumenten, welche jedoch unterschiedlich datiert sind und dadurch verschiedene Bezeichnungen für das System und dessen Komponenten verwenden.

3.2.4 Zusammenfassung

In dieser Arbeit soll ein Weg gefunden werden, wie die Integration einer Geschäftsanwendung in verschiedene Single Sign-On Systeme funktionieren kann. Dabei alle am Markt befindlichen Lösungen zu berücksichtigen, würde den zeitlichen Rahmen sowie den Umfang sprengen.

CAS zeichnet sich durch seine Leichtgewichtigkeit aus, da es nur die Authentisierung behandelt. Es bietet Sicherheit durch die Verwendung von HTTPS und die Arbeitsweise ähnelt der von Kerberos. JOSSO unterstützt Windows-Authentisierung, basiert ebenso wie CAS auf dem Spring Framework und wird über XML-Dateien konfiguriert. OpenSSO dagegen besitzt eine umfassende Administrationskonsole und eine Vielzahl von Komponenten sowie ein umfangreiches Client SDK.

Die folgende Tabelle zeigt eine Auswahl der Eigenschaften und Funktionalitäten der bereits beschriebenen SSO-Systeme. Die Lösungen weisen in einigen Bereichen Gemeinsamkeiten auf. So ist der Aufbau mit Server und Client in allen Systemen ähnlich. Als Nutzerdatenspeicher wird von allen drei Systemen LDAP unterstützt. Für die Anmeldung am BLS können verschiedene Authentisierungsmethoden verwendet werden.

	CAS	JOSSO	OpenSSO
APIs für Clients / Agents	u.a. JEE, .NET, PHP, Perl, Apache	JEE, .NET, PHP, Python, Perl	JEE, IIS[14], SAP, OAS[15]
Architektur	zentral	zentral	zentral, Verbund
	CAS Server, CASified Clients	JOSSO Agent (SP), Gateway (IdP)	Access Manager,, Policy Agent
Authentisierung	LDAP, DB, Remember-Me, Generic, X.509, JAAS	LDAP, DB, Remember-Me, Strong, NTLM	LDAP, AD, JDBC, Windows Desktop SSO, Unix, X.509
Administration	XML-Konf.-Dateien	XML-Konf.-Dateien	Text- und Webkonsole
Autorisierung	-	Spring Security	SAML
Nutzerdatenspeicher	LDAP, JDBC	LDAP, DB (JDBC)	u.a. LDAP, Active Directory
Enterprise-SSO	√	√	√
Server Komponente	Tomcat	Tomcat, JBoss, WebLogic, WebShere	Tomcat, JBoss, WebLogic, WebSphere, Sun App. Server
Sessionbehandlung	Service Tickets, Ticket Granting Cookies	AuthenticationAssertions	SSOTokens
Single Log-Out	√	√	√
Spring Security Unterstützung	√	√	√ (als Extension)
Version	3.3.5 (11/09)	1.8.1 (09/09)	8.0 Update 1 (05/09)
Web-basiert	√	√	√

Abb. 3-8: Übersicht SSO-Systeme. [Stand: 11/2009]

Auswahl

Bei der Entscheidung für eine der drei SSO-Lösungen bleiben lizenzrechtliche Bedenken außen vor, da alle vorgestellten Systeme als Open-Source-Produkte frei erhältlich sind.

14 IIS: Internet Information Server

15 OAS: Oracle Information Server

Die betrachteten Systeme sind auf Webanwendungen ausgerichtet, also solche, die in einem Webbrowser laufen. Der zu entwickelnde Integrations-Prototyp soll jedoch auch für Desktop-Anwendungen nutzbar sein. OpenSSO ist die einzige Lösung, die ausdrücklich Unterstützung für Non-Web-Anwendungen bietet. Das sind Programme, die zwar mit Serverkomponenten kommunizieren können, aber nicht im Browser, sondern auf dem dem Desktop ausgeführt werden.

Grundsätzlich sollte die Integration des Leasman Business Logic Server in alle Single Sign-On Systeme möglich sein, die verteilte Anwendungen unterstützen. Aus den hier Vorgestellten wurde OpenSSO ausgewählt, da es die umfangreichste Dokumentation und durch das *ClientSDK* eine gute Programmierbasis besitzt.

Um zu demonstrieren, dass die Geschäftsanwendung mit Hilfe des zu entwickelnden Prototypen in verschiedene SSO-Lösungen integriert werden kann, wird für ein weiteres System eine Implementierung angestrebt.

Die Auswahl ist nicht exklusiv zu verstehen, da die Systeme nicht als Konkurrenten sondern vielmehr als gleichberechtigte Probanden betrachtet werden. Es soll hier nicht etwa die „beste“ Lösung herausgestellt werden.

Weitere vergleichende Übersichten sind „Vergleich von Java SSO Lösungen“ [ssv] und „Fast and Free SSO“ [oso].

4 Konzeption

Der Entwurf des Prototypen erfolgt in drei Schritten. Ein erstes Grobkonzept beschreibt Anforderungen, die erfüllt werden müssen oder wünschenswert sind. Im darauf folgenden Abschnitt werden die gestellten Anforderungen in Verbindung mit Erkenntnissen über SSO-Systeme in der Definition einer Schnittstelle zusammengeführt. Das anschließende Szenario beschreibt die geplante SSO-Integration der Geschäftsanwendung anhand der entworfenen Schnittstelle.

4.1 Anforderungen

Der zu erstellende Prototyp stellt ein Paket von Komponenten dar, welche zusammengenommen die Funktionalität bieten, die vorhandene Geschäftsanwendung in Single Sign-On Umgebungen einzubinden.

Die Implementierung soll dabei vorwiegend auf der Serverseite stattfinden, um Änderungen an den Clients minimal zu halten und diese zu entlasten. Dadurch wird auch das Bestreben unterstützt, in der Entwicklung hauptsächlich Java zu verwenden und so wenig wie möglich Änderungen mit PowerBuilder, der Entwicklungsumgebung für den Leasman Client vorzunehmen.

Es ist zu gewährleisten, dass der Prototyp den Anforderungen an Portier- und Erweiterbarkeit genügt. Die Integration in ein anderes als das hier exemplarisch verwendete SSO-System soll mit möglichst wenig Aufwand verbunden sein. Dafür ist eine hohe Abstraktionsstufe notwendig.

Weiterhin muss die Sicherheit bezüglich sensibler Nutzerdaten und Kommunikationsvorgängen grundlegend gewährt werden. Zudem ist eine hinreichende Test- und Wartbarkeit zu berücksichtigen.

Client-Anwendungen, welche die Geschäftslogik des BLS nutzen, können auf unterschiedlichen Plattformen laufen, da sie systemunabhängig, beispielsweise Web Service-basiert mit der Serverkomponente kommunizieren. Wünschenswert wäre die Möglichkeit der Anmeldung über die Nutzerkennung des Systems, auf welchem der Client ausgeführt wird. In diesem Punkt müssen jedoch die jeweiligen Anmelde- und Nutzerverwaltungsverfahren der Betriebssysteme beachtet werden, wodurch sich voraussichtlich der Aufwand der Umsetzung in wesentlichem Maße erhöht.

Bei der vollständigen Integration in ein Single Sign-On System könnte dieses sowohl Authentication-, Authorization-, Identity- als auch Sessionmanagement übernehmen. Der Fokus des Prototypen liegt auf der Authentisierung, wobei auf die übrigen Bereiche, speziell Sitzungen und Identitäten, wenigstens grundlegend eingegangen werden muss. Die Verwaltung und Behandlung von Rechten verbleibt bei der Anwendung und wird nicht bearbeitet.

Als Ergebnis soll der Business Logic Server befähigt werden, die SSO-Systemintegration als Option anzubieten.

4.2 Schnittstellendefinition

Als Ziel dieser Arbeit soll unter anderem eine Möglichkeit gezeigt werden, wie die beschriebene Geschäftsanwendung in SSO-Systeme integriert werden kann. Obwohl die vorgestellten Systeme zum Teil sehr unterschiedlich aufgebaut sind, weisen sie doch grundsätzlich gemeinsame Funktionalitäten auf. Sie sind in der Lage, Authentisierungsvorgänge über Tickets zu realisieren. Zudem können sie Benutzeridentitäten verwalten oder bieten Schnittstellen zu Identity-Management-Lösungen. Diese Fähigkeiten werden benötigt, um das Authentisierungs-Szenario aus 2.2.3 in Verbindung mit dem in 3.1.6 genannten Ansatz für ticketbasiertes SSO zu realisieren.

Die Gemeinsamkeiten können nun als Basis für eine **Abstraktion** dienen, die es ermöglicht, die Fähigkeiten von SSO-Systemen in der Geschäftsanwendung zu nutzen, ohne jedoch an ein spezifisches System gebunden zu sein.

Java *Interfaces*[16] bieten die Möglichkeit, diese Abstraktion als Konstrukt aus Eigenschaften – in Java *Fields*[17] genannt – und Methoden programmiertechnisch abzubilden. Dabei können die Felder bereits vordefinierte Werte haben, wohingegen für die Methoden nur die Signatur mit Namen und eventuellen Parametern festgelegt wird. Die Methodenrümpfe bleiben leer und werden durch die entsprechende Implementierungsklasse mit Funktionalität gefüllt.

4.2.1 Namenskonventionen

Um Interfaces als solche zu kennzeichnen, wird dem Namen ein „`I`“ vorangestellt. Die angestrebte Schnittstelle bietet Funktionen zur Integration in Single Sign-On Systeme an. Davon abgeleitet ergibt sich „`ISSOService`“ als Namensentwurf.

In Java werden Klassen und Schnittstellen in *Packages* organisiert. Diese Pakete lassen sich zusammengenommen als hierarchische Baumstruktur abbilden, ähnlich wie die Verzeichnisse eines Dateisystems. Beim Verwenden einer Klasse aus einem be-

16 dt. Schnittstellen

17 dt. Felder

stimmten Paket wird sie über die Paketnamen, von der obersten Ebene absteigend und jeweils getrennt durch einen Punkt, angesprochen. Der Klassenname in Verbindung mit diesem Präfix wird auch als *fully qualified class name* bezeichnet.

Die Namensgebung orientiert sich häufig an vorhandenen Organisationsstrukturen. So auch bei der DELTA proveris AG. Paketnamen beginnen mit „`de.delta`“ . Daran anknüpfend wird für die Schnittstelle ein Paket „`sso`“ angelegt, sodass sich „`de.delta.sso.ISSOService`“ als qualifizierter Name für die Schnittstelle ergibt.

4.2.2 Abbildung eines Credential-Objektes

Die Geschäftsanwendung verwendet die Schnittstelle, um Informationen aus den Tickets des SSO-Systems auslesen zu können. Tickets werden für die Einmalanmeldung gegenüber Anwendungen benötigt.

Um eine hinreichende Abstraktion zu erhalten, wird „Credential“ als Begriff verwendet, um ein Objekt zu bezeichnen, das in SSO-Systemen unterschiedlich benannt wird, wie etwa „Token“ oder „Ticket“. Es handelt sich also um ein Objekt, das einer bestimmten Identität, zum Beispiel einer Person zugeordnet ist, bestimmten zeitlichen Begrenzungen unterliegt und weitere beliebige Eigenschaften haben kann.

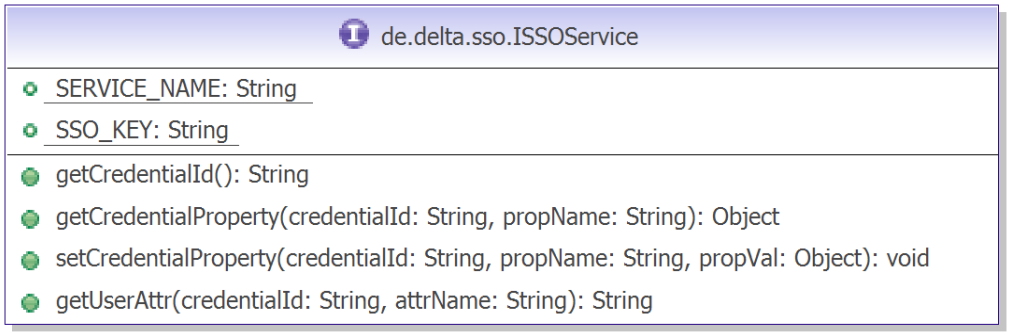


Abb. 4-1: ISSOService UML

Das wichtigste Attribut dieses Credential-Objektes ist seine ID[18], mit der es eindeutig angesprochen werden kann. Um ein solches Objekt in der Schnittstelle zu repräsentieren, bietet sie Methoden zum Auslesen der ID via `getCredentialId()`, dem Setzen beziehungsweise Abfragen von Eigenschaften durch `set-` und `getCredentialProperty()` und dem Auslesen von Attributen des Credential-Besitzers durch `getUserAttr()`.

Die Credential Objekte werden zustandslos verwendet, um ungewollte Kopplungen und Abhängigkeiten zu vermeiden. Die Objekte werden nicht als Felder der SSO-Service-Klasse gespeichert, sondern bei jeder Verwendung über die Credential-ID identifiziert und angesprochen. So bleiben sie Repräsentatoren der zugehörigen Sessions, die vom jeweiligen SSO-Server verwaltet werden.

18 ID: Identifikator, Identifikationskennzeichen, -merkmal

4.3 Integrationsszenario

In Single Sign-On Systeme integriert können Anwendungen Tickets zur Authentisierung verwenden, um die Geschäftslogik des Business Logic Servers zu nutzen. Dadurch entfallen für den Nutzer die interaktiven Anmeldevorgänge nach der Primärauthentisierung. Dabei wird ein zentraler Single Sign-On Server benötigt, der Zugriff auf die Nutzerdaten hat und über den eine SSO-Session initiiert wird.

Mit Clients sind im Folgenden Anwendungen gemeint, die mit Hilfe von Java-Client Bibliotheken die EJB-Schnittstelle des BLS nutzen können.

4.3.1 Anmeldevorgang am BLS

Bei der Verbindung von Client Anwendungen zum Business Logic Server werden einige Funktionalitäten wie beispielsweise Session- und Fehlerbehandlung über Interceptoren realisiert. Als Einstiegspunkt für die SSO-Integration wird diese Kette von Interceptoren gewählt, die im BLS durch die Konfigurationsdatei `chain-of-request.xml` abgebildet wird. Für die geplante Integration werden sowohl client- als auch serverseitig Interceptor-Klassen durch die Verwendung der in 4.2 definierten SSO-Schnittstelle entweder erweitert oder neu erstellt.

Client Anwendungen werden über bestehende Schnittstellen an den Business Logic Server angebunden. Die Clients rufen die BLS-Geschäftslogik auf. Ein Kommandotransportobjekt (CommandTO) kapselt den Aufruf der Geschäftslogikmethoden. Es transportiert alle Informationen, die zur Benutzung des speziellen Service notwendig sind. Das CommandTO enthält den Namen der Komponente und deren Methoden, die aufgerufen werden sollen, sowie die notwendigen Parameter. Zusätzlich sind noch sogenannte Header-Parameter enthalten, die unabhängig von der Komponente verwendet werden [bdo]. Dieser Vorgehensweise liegt das *Command Pattern*[19] zugrunde. Danach lassen sich Requests eines Senders an einen Empfänger als Objekte behandeln. Der Sender wird vom Empfänger entkoppelt, sodass er des-

19 ein Entwurfsmuster

sen Schnittstelle nicht kennen muss. Mit dieser Entwurfsstrategie lässt sich bestimmen, wann und wie eine Aktion ausgeführt werden soll [cmp].

Im vorliegenden Abschnitt wird synonym zur bereits genannten Bezeichnung „Credential“ der Begriff „Ticket“ gebraucht, um die Anschaulichkeit zu verbessern. Wichtig für das Verständnis ist die Unterscheidung der Session-ID und der ID des Tickets. Beide identifizieren eine Session. Die erste, in dieser Arbeit auch als BLS Session-ID bezeichnet, bezieht sich auf die Sitzung, die der Anmeldung am BLS folgt. Das Ticket repräsentiert eine Single Sign-On Session, die durch den SSO-Server verwaltet wird und unabhängig vom BLS läuft. Die Ticket-ID wird verwendet, um Daten im Zusammenhang mit der SSO-Session abzufragen oder zu speichern. Bei diesen Vorgängen kommt jeweils das SSOService-Interface zum Einsatz.

Client

In einem Interceptor auf der Clientseite wird unter Nutzung des SSOService ein Ticket an das zuständige Transportobjekt, das CommandTO, angehängt. Genauer gesagt wird, wie in der Schnittstellendefinition beschrieben, die ID des Tickets verwendet. Wenn bereits ein gültiges Ticket vorhanden ist, soll dieses verwendet, andernfalls ein Neues erzeugt werden. Anschließend werden die in `chain-of-request.xml` benannten weiteren Interceptoren durchlaufen.

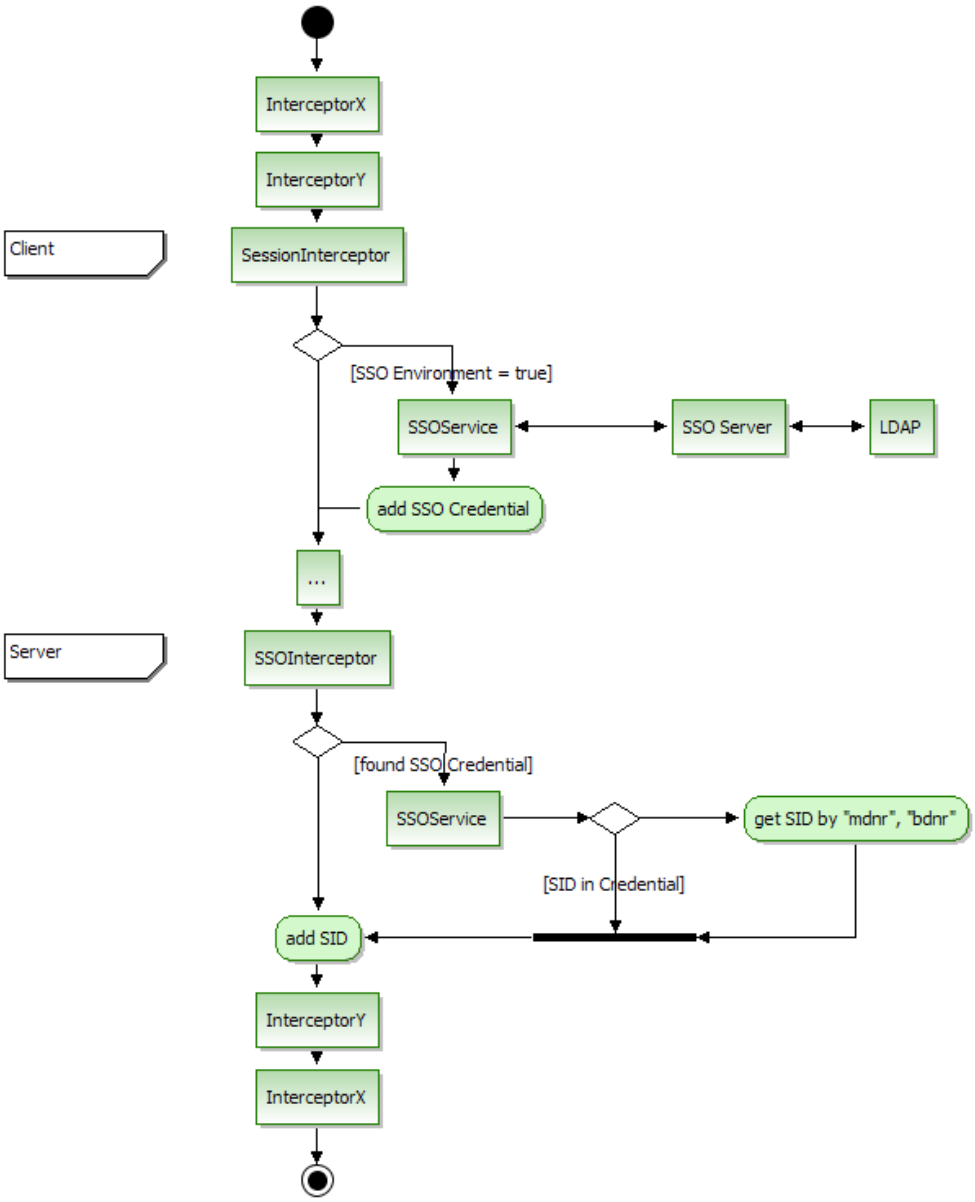


Abb. 4-2: SSOService in der Interceptor-Kette

Server

Auf der Serverseite wird als Kontrollinstanz ein Interceptor benötigt, der das Vorhandensein eines Tickets prüft und daraufhin die Anmeldung übernimmt. Für die Anmeldung am BLS ist eine Session-ID nötig. Um diese zu erhalten, sind die folgenden Schritte notwendig:

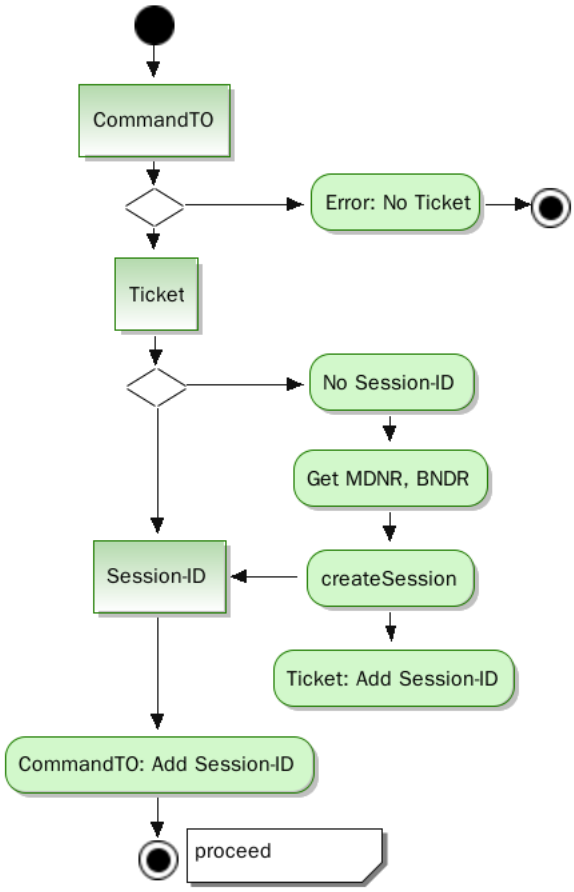


Abb. 4-3: BLS Session-ID aus Ticket

Im ersten Schritt wird aus dem Header des Kommandotransportobjektes die ID eines Tickets geholt. Ist keine vorhanden, erfolgt ein Abbruch mit einer Fehlermeldung. Da es sich um eine SSO-Umgebung handelt, kann ohne das Ticket nicht fortgesetzt werden.

Wenn das Ticket vorhanden ist, jedoch keine BLS Session-ID (SID) enthält, wird die Methode `createSession(mdnr, bdnr)` des BLS SessionService aufgerufen. Dieser Methode werden zwei Parameter übergeben: Die Nummer des Leasman Mandan-

ten, für den die Session initiiert wird und die Leasman Bedienernummer, unter der alle nachfolgenden Aktionen der Session durchgeführt werden. Die durch `createSession` erstellte Sitzung wird für das Setzen von Objektsperren beim Bearbeiten von Geschäftsobjekten benötigt. Die Methode gibt die Session-ID zurück, die bei allen nachfolgenden Methodenaufrufen als Header-Parameter im CommandTO angegeben werden muss. [bdo]

Das CommandTO benötigt die SID für das erfolgreiche Durchlaufen der „Request-Kette". Außerdem wird sie als Property im Ticket gespeichert, um beim nächsten Start abrufbar zu sein. Durch die abschließende `proceed`-Methode wird an den Punkt zurückgesprungen, an dem der Interceptor aufgerufen wurde.

4.3.2 Konfiguration

Folgende Anpassungen und Erweiterungen der vorhandenen BLS Auslieferung sind für die SSO-Integration erforderlich:

- ISSOService und verwendete Implementierungsklasse sowie gegebenenfalls weitere benötigte Ressourcen zur Verfügung stellen (als JAR in Classpath)
- Jeweils Bean-Definition für ISSOService mit Angabe der Implementierung in Konfigurationsdatei auf Client- und Serverseite; bei Angabe der qualifizierten Namen auf Übereinstimmung mit Packages achten
- Interceptor auf Client- und Serverseite als Java Klasse hinzufügen bzw. anpassen
- Jeweils Bean-Definition für Interceptor per XML-Konfiguration und Einfügen in bestehende *chain-of-request*-Liste

Voraussetzung ist ein zur Implementierung passender SSO-Server, der in einem Application Server installiert ist. Die Verbindung zu diesem wird in einer systemspezifischen Datei eingestellt.

Wenn nicht bereits durch die SSO-Lösung bereitgestellt, wird außerdem ein Directory Server als Nutzerdatenspeicher benötigt. Dort muss ein Eintrag zu einer Person mit den Attributen „mdnr“ und „bdnr“ existieren. Der Distinguished Name (DN) dieser Person, der sie von anderen Einträgen im LDAP-Kontext unterscheidet, wird zur eindeutigen Referenzierung in einer Konfigurationsdatei hinterlegt.

5 Implementierung

In diesem Kapitel wird nach einer anfänglichen Beschreibung über Art und Umfang der prototypischen Implementierung erläutert, wie die SSO-Integration der Geschäftsanwendung durch die Schnittstelle umgesetzt wurde. Es folgen Ausführungen über die Implementierungen der Schnittstelle und die Konfiguration der SSO-Umgebungen. Schließlich wird anhand eines Unit-Tests die Funktionsfähigkeit des Prototypen demonstriert. Der letzte Abschnitt enthält eine Problembeschreibung.

5.1 Überblick

Durch die Implementierung wird ein abstraktes Konstrukt zu einem greifbaren Stück Software, das ein eigenständig lauffähiges Programm sein kann oder als Teil einer bereits bestehenden Anwendung diese durch Zusatzfunktionalität erweitert. Bei der Umsetzung können Anpassungen zur Integration in die Zielumgebung notwendig werden.

Ein Software-Prototyp will Lösungen demonstrieren und dient zur Beurteilung der Machbarkeit von Anforderungen und Entwürfen. So können frühzeitig Probleme erkannt und korrigiert werden.

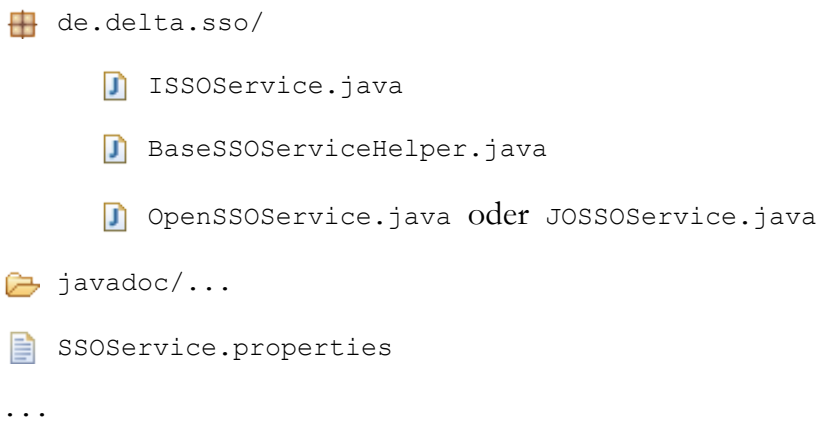
> *„Mit einem Prototypen streben Sie an, bestimmte Aspekte des endgültigen Systems zu erkunden. Bei einem echten Prototypen werden Sie alles, was Sie bei Ihren Versuchen zusammengestrickt haben, wegwerfen und es mit den gewonnenen Erfahrungen noch mal programmieren.“ [prg03] S.47*

Die prototypische Implementierung der in Kapitel 4 entworfenen Schnittstelle wurde für zwei freie Single Sign-On Systeme umgesetzt. Hinsichtlich der Einheitlichkeit und Wiedererkennbarkeit wurden, ausgehend von `SSOService` als Wortstamm, die entstandenen Klassen `OpenSSOService` und `JOSSOService` benannt. Dabei wurde

auch eine Helferklasse `BaseSSOServiceHelper` erzeugt, die grundlegende Funktionalität unabhängig vom verwendeten SSO-System bietet. BaseSSOServiceHelper stellt die Basisklasse für OpenSSOService und JOSSOService dar, das heißt, die beiden Implementierungen der SSOService-Schnittstelle erben die Eigenschaften (Felder) und Methoden der Helferklasse. Bereitgestellt werden durch diese Klasse Funktionen zum Auslesen von `.properties`-Dateien sowie ein Feld, das den Subklassen die Fähigkeit des *Logging* verleiht.

Aufbau des fertigen Paketes

Die Bestandteile des Prototyps sind in einem Java-Archiv (JAR) zusammengefasst, welches dem Classpath der BLS-Sourcen hinzugefügt werden kann. Das Archiv ist wie folgt aufgebaut:



Das Paket `de.delta.sso` enthält das SSOService-Interface, die Helferklasse `BaseSSOServiceHelper` und die jeweilige Implementierung der Service-Schnittstelle, also `OpenSSOService` oder `JOSSOService`. Das JAR liefert weiterhin die Datei `SSOService.properties`, in der Einstellungen zum Benutzer vorgenommen werden sowie einen Ordner `javadoc` mit der HTML-basierten API-Dokumentation. Je nach Implementierung werden weitere Pakete und Konfigurationsdateien ausgeliefert.

Für die Auslieferung des BLS mit der Option zur SSO-Integration werden die genannten Teile über die Entwicklungsumgebung eingebunden und der BLS wird als Enterprise Archive (EAR) zusammen mit Batch- und Konfigurationsdateien ausgeliefert.

Der Quelltext der entwickelten Java-Klassen kann im Anhang nachvollzogen werden. Die Dateien sind zum besseren Verständnis mit Bemerkungen in Form von Javadoc-Kommentaren versehen.

5.2 Integration der Geschäftsanwendung

Client

Clientseitig wurde der `SessionInterceptor` dahingehend erweitert, dass dem Kommandotransportobjekt über die Methode `addHeaderParam()` ein String angehängt wird, der die ID einer Single Sign-On Session repräsentiert. Die zugehörige Funktion des SSOService, die diese Zeichenkette zurück gibt, ist `getCredentialId()`. Bei der Angabe des Namens für den Header-Parameter wird mit `SSOService.SSO_KEY` ein String benutzt, den das Interface zur Identifikation der Credential-IDs bereitstellt.

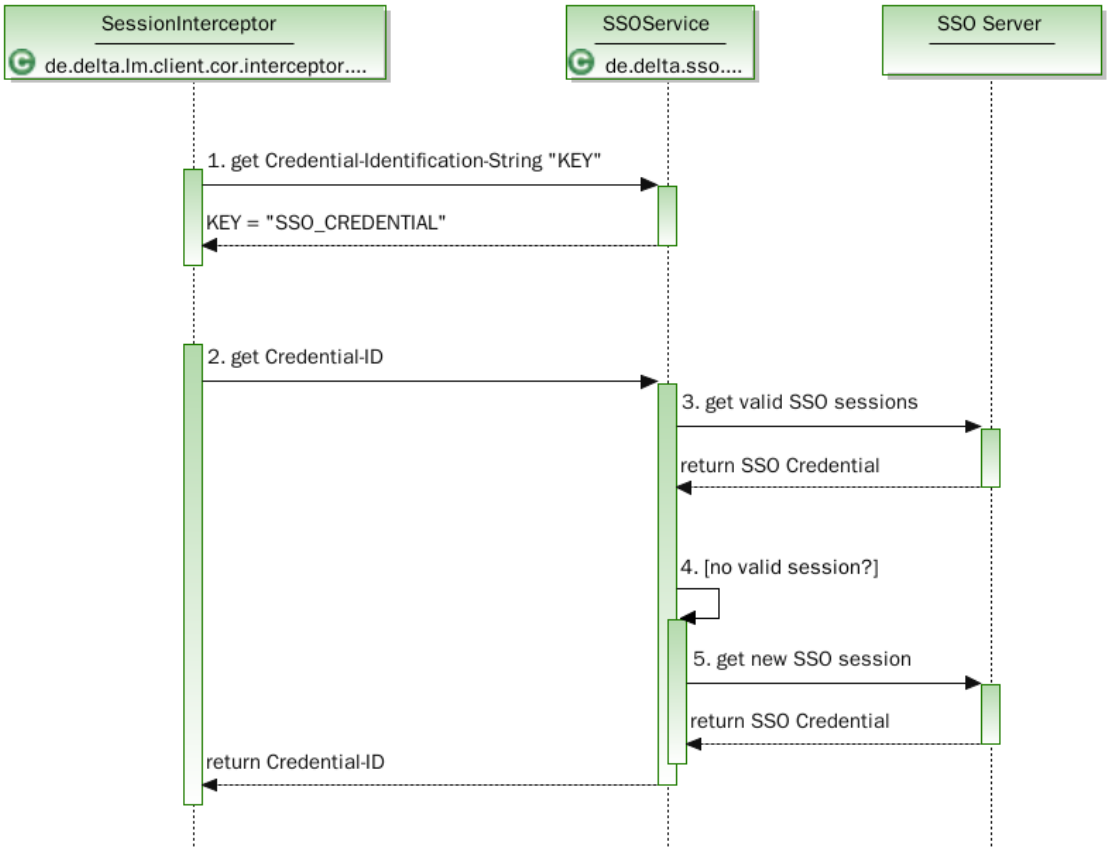


Abb. 5-1: SSOService clientseitig

Der Codeausschnitt zeigt die Verwendung des SSOService im SessionInterceptor:

```
@Autowired
@Qualifier("session.ISSOService")
private ISSOService m_ssoService;

public Object invoke(MethodInvocation p_arg0) throws Throwable
{
 CommandTO cmd = (CommandTO)p_arg0.getArguments()[0];

 if (cmd.getHeaderParam(ISSOService.SSO_KEY) == null) {
         cmd.addHeaderParam(
               ISSOService.SSO_KEY,
               m_ssoService.getCredentialId()
         );
 }
```

Im SessionInterceptor wird die `getCredentialId`-Methode des SSOService-Interface aufgerufen. Auf welche Weise die Credential-ID geholt oder erzeugt wird, bestimmt die jeweilige Implementierung.

> *„Das Programmieren gegen Interfaces statt direkte Implementierungen ist sowieso eine gute Idee. Dadurch hängen die Objekte nur noch von den Interfaces ab und nicht mehr von den Klassen, so dass man leichter die konkrete Implementierung austauschen kann. Nachteil ist natürlich das zusätzliche Interface.“ [spr] S. 19*

Der SSOService wird mittels Annotation, erkennbar durch das “`@`”, als Java Bean verwendet. Gleiches geschieht serverseitig im SSOInterceptor. In beiden Fällen muss die Bean in entsprechenden Konfigurationsdateien bekannt gemacht werden, so etwa in der `chain-of-request.xml` des Client:

```
<bean name="session.ISSOService"
    class="de.delta.sso.OpenSSOService">
</bean>
```

In `class` steht dann jeweils der qualifizierte Name der Klasse, die für die Implementierung des SSOService verwendet werden soll.

Server

Das Einhängen des SSOInterceptors erfolgt in der Datei `chain-of-request.xml`, indem zunächst für die Interceptor-Klasse eine Bean definiert und dann in die Liste der Interceptoren des `TargetInvocationAdapter` eingefügt wird.

```
...
<bean id="interceptor.SSOInterceptor"
  class="de.delta.lm.server.cor.interceptor.SSOInterceptor">
</bean>
...
<bean name="TargetInvocationAdapter"
   class="org.springframework.aop.framework.ProxyFactoryBean">
...
 <property name="target">
    <ref bean="target.TargetInvocationAdapter"/>
 </property>
 <property name="interceptorNames">
    <list>
         <value>interceptor.ExceptionInterceptor</value>
         <value>interceptor.SSOInterceptor</value>
         <value>interceptor.BundleInterceptor</value>
         <value>interceptor.SessionInterceptor</value>
         <value>interceptor.RemovableLocksInterceptor</value>
    </list>
 </property>
</bean>
```

Der SSOInterceptor liest die Credential-ID als Header-Parameter aus dem CommandTO aus. Falls keine solche ID vorhanden ist, wird eine Exception (Ausnahme) geworfen. Es wird hier davon ausgegangen, dass der SSOInterceptor nur eingebaut ist, wenn der BLS in einer Single Sign-On Umgebung läuft. Daher wird an dieser Stelle ein „Ticket“ gefordert. Ist dieses nicht vorhanden, kann die Anmeldung nicht vollzogen werden.

Die ausgelesene ID wird verwendet, um die im zugehörigen Credential enthaltene BLS Session-ID durch Verwendung der SSOService-Methode `getCredentialProperty()` zu reproduzieren. Wenn sie nicht da ist, wird sie über die Methode `createSession()` des BLS SessionService geholt. `CreateSession()` benötigt die Mandanten- und Bedienernummer des Credential-Besitzers. Diese beiden Nummern sind zu Testzwecken im Datenspeicher eines Nutzers namens „leasmantester“ im OpenLDAP hinterlegt. Der SSOService liest die Werte dieser beiden Attribute mit `getUserAttr()` aus und der SessionService holt die zugehörige Session-ID. Anschließend wird diese direkt per `setCredentialProperty()` an das Credential angehängt und ist beim nächsten Aufruf verfügbar.

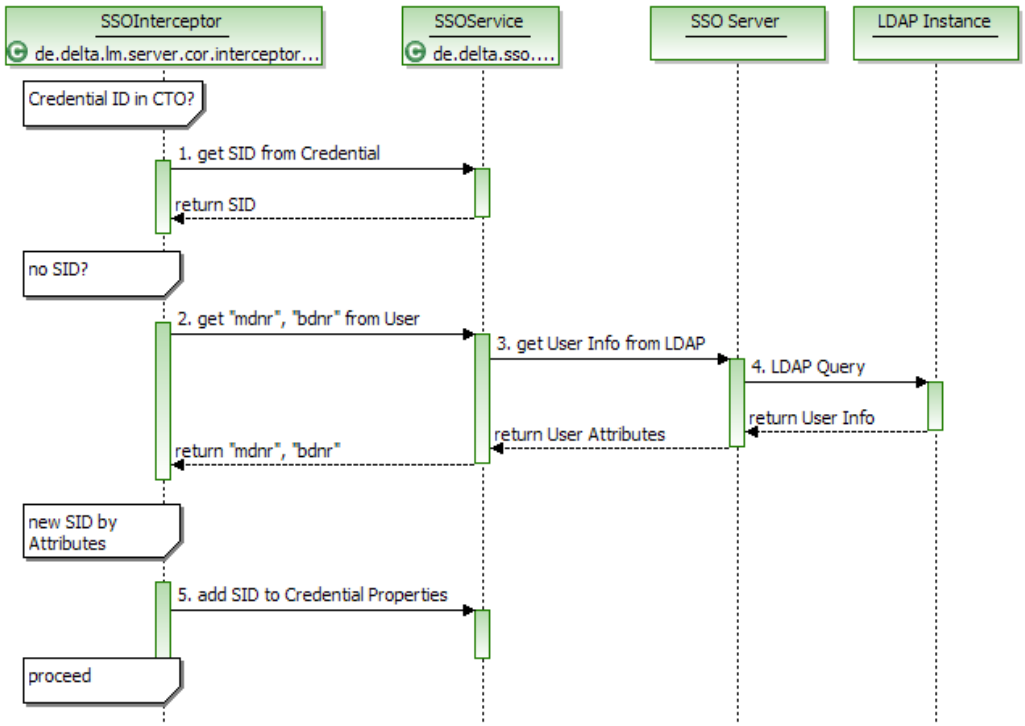


Abb. 5-2: SSOService serverseitig

Der `SERVICE_NAME` des SSOService wird verwendet, um einen Namen für die Schnittstelle festzulegen. Für die Konfiguration existiert eine Datei `SSOService.properties`, aus welcher die Service-Implementierung die zu verwendenden Nutzerdaten zur Authentifizierung bezieht. Zur eindeutigen Zuordnung der Datei zum Service entspricht deren Name ohne Dateiendung dem `SERVICE_NAME`.

5.3 Implementierung für OpenSSO

Die Implementierung des SSOService wurde mit Hilfe des Client SDK (Software Development Kit) von OpenSSO umgesetzt. Über eine Instanz von `SSOTokenManager` werden Credentials in Form von `SSOTokens` erstellt. Die Authentisierung erfolgt in der Methode `getSessionToken()` durch Verwendung der Klasse `AuthContext`, die Nutzername und Passwort gegenüber dem SSO-Server validiert. Die SSO-Tokens bieten die Möglichkeit, eigene Properties im Token zu speichern.

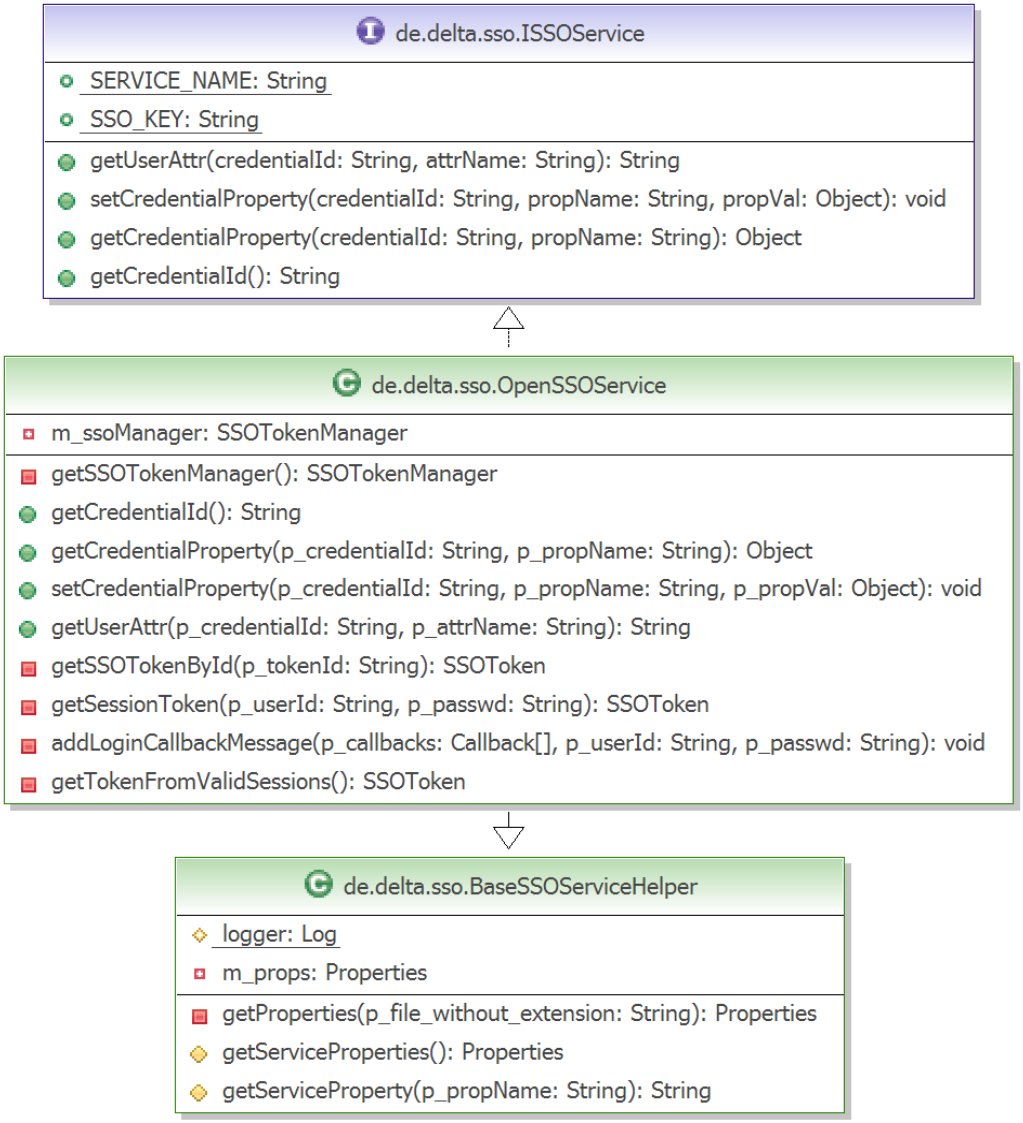


Abb. 5-3: OpenSSOService UML

Im Data Store hinterlegte Attribute des Nutzers können durch `AMIdentity.getAttribute()` ausgelesen werden.

Es wird eine Konfigurationsdatei namens `AMConfig.properties` benötigt, um Einstellungen bezüglich der Verbindung zum OpenSSO-Server vorzunehmen. Fehlt diese Datei, kann der Server nicht gefunden und somit die SSO-Funktionalität nicht genutzt werden.

An dieser Stelle erfolgt eine knappe Beschreibung der Schritte, die zum lauffähigen SSO-Server und Nutzerdatenspeicher führen. Ausführliche Dokumentationen zu den verwendeten Systemen finden sich unter anderem bei GlassFish[20], OpenSSO[21] und OpenLDAP[22].

5.3.1 Application Server

Der von OpenSSO bevorzugte *Deployment Container*[23] ist der *Sun GlassFish Enterprise Server*. Für die vorliegende Implementierung wurde er in der Version 2.1 verwendet.

```
root@woteba:~# sso
# STARTING OPENSSO DOMAIN...
Domäne mydomain wird gestartet, bitte warten.
Default Log location is /homes/delta/pem/glassfish/domains/mydomain/logs/server.log.
Ausgabe wird umgeleitet in /homes/delta/pem/glassfish/domains/mydomain/logs/server.log
Domäne mydomain ist bereit zum Empfangen von Clientanforderungen. Zusätzliche Dienste werden
 im Hintergrund gestartet.
In Domäne [mydomain] wird [Sun GlassFish Enterprise Server v2.1 (9.1.1) (build b60e-fcs)] au
sgeführt; die zugehörige Konfiguration und die zugehörigen Protokolle befinden sich hier: [/
homes/delta/pem/glassfish/domains].
Admin-Konsole ist unter [http://localhost:4848] verfügbar.
Verwenden Sie denselben Port [4848] für "asadmin"-Befehle.
Benutzerwebanwendungen sind unter folgenden URLs verfügbar:
[http://localhost:8080 https://localhost:8181 ].
Folgende web-contexts sind verfügbar:
[/web1  /__wstx-services opensso opensso-client-jdk15 cas-server-webapp-3 josso-gateway fedl
et StockService StockClient jguard-struts-example ].
Standardmäßige JMX-Clients (etwa JConsole) können eine Verbindung mit JMXServiceURL herstell
en:
[service:jmx:rmi:///jndi/rmi://woteba:8686/jmxrmi] für Domänenverwaltungszwecke.
Domäne hört zumindest folgende Ports auf Verbindungen ab:
[8080 8181 4848 3700 3820 3920 8686 ].
Die Domäne unterstützt keine Application Server-Cluster und keine anderen eigenständigen Ins
tanzen.
```

Abb. 5-4: OpenSSO-Server per SSH Konsole starten

20 siehe http://docs.sun.com/app/docs/coll/1343.11

21 siehe http://developers.sun.com/identity/reference/techart/opensso-glassfish.html

22 siehe http://wiki.ubuntu-forum.de/index.php/OpenLDAP, bzw. [ldp03]

23 Ein Application Server mit der Fähigkeit, Archivdateien wie WAR oder EAR als (Web-)Anwendung bereitzustellen

GlassFish wurde auf einem virtuellen Ubuntu Server 8.04 installiert. Die Arbeit auf diesem Server erfolgte über eine SSH-Konsole. Im Application Server musste eine Domain angelegt werden, der bestimmte Ports für die Administrationskonsole sowie verschiedene Benutzeranwendungen zugeordnet sind.

Das Deployen[24] des Webarchives `opensso.war` kann auf verschiedenen Wegen geschehen. Einer davon führt über die webbasierte Administration der im GlassFish angelegten Domain (Abb. 5-5). Hier kann über die Auswahl der WAR-Datei und mit zusätzlichen Einstellmöglichkeiten das Archiv deployt werden.

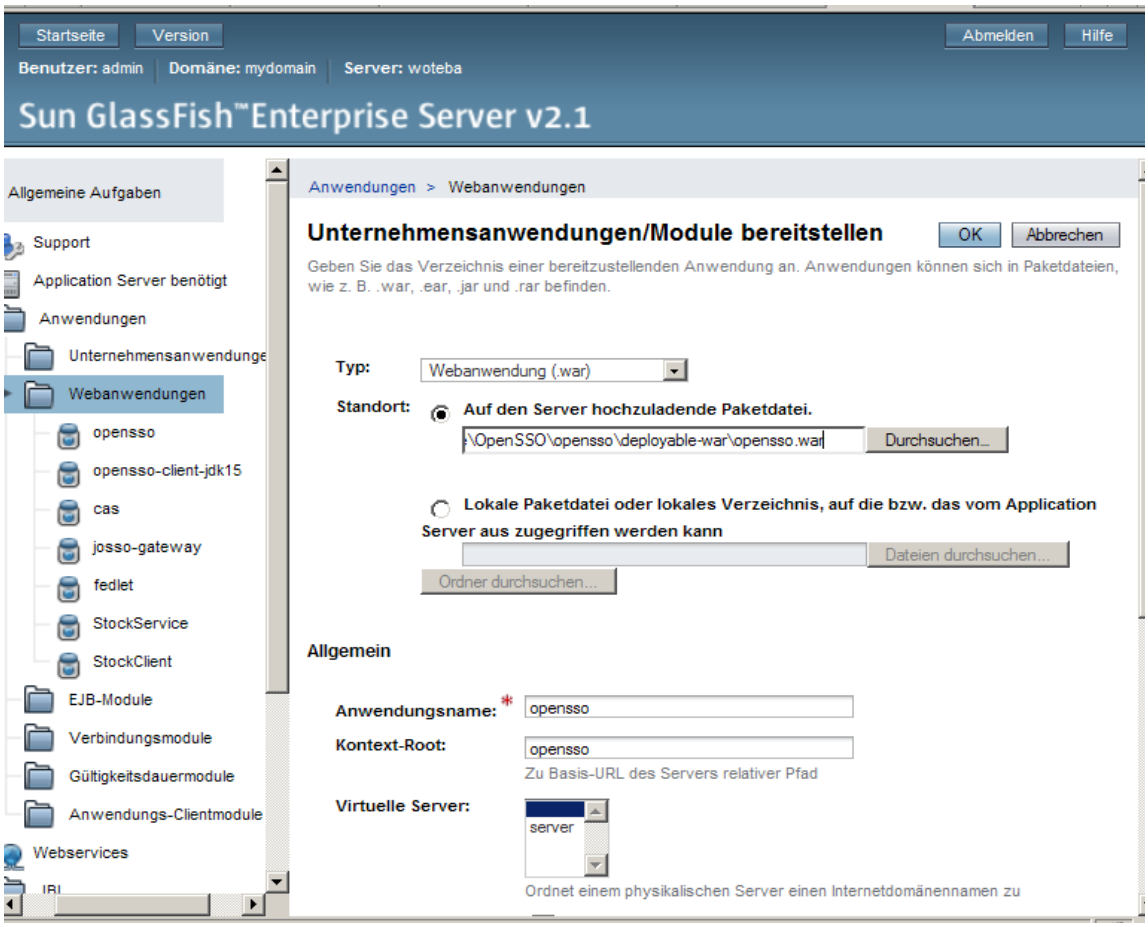


Abb. 5-5: Webarchiv im GlassFish bereitstellen

Anschließend ist auf dem Server unter der URL mit dem vorher angegebenen Namen des *Deployment Descriptors* ein Installationsassistent zu erreichen. Der Deploy-

24 dt. bereitstellen, installieren

ment Descriptor entspricht per default dem Namen des Webarchives ohne Dateiendung und ist in Abbildung 5-5 als Anwendungsname zu finden.

Bei Auswahl der benutzerdefinierten Konfiguration werden Einstellungen zum Administrator, der Konfigurations- und der Nutzerdatenbank getätigt. Nachdem die Installation abgeschlossen ist, wird zur Administrationskonsole (Abb. 5-6) von OpenSSO weitergeleitet, an der man sich mit dem zuvor vergebenen Admin Account anmeldet.

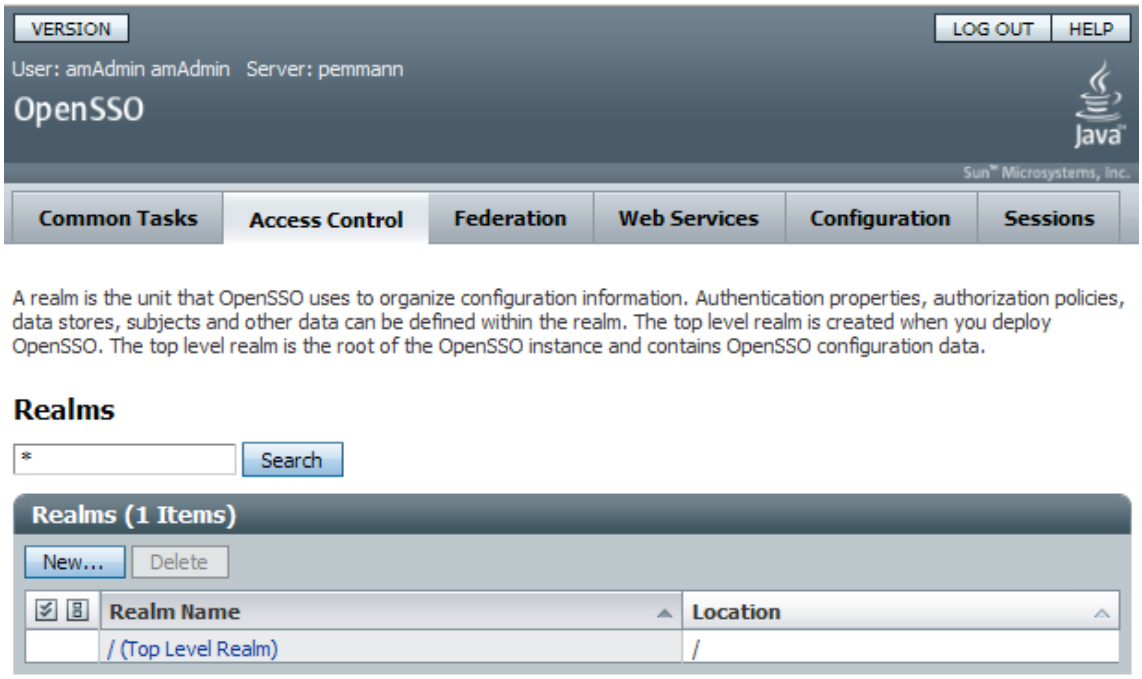


Abb. 5-6: OpenSSO Admin Console

Über diese webbasierte Konsole können viele administrative Arbeiten durchgeführt werden. Sogenannte Realms[25] können angelegt und verwaltet werden. Diesen Realms sind wiederum Authentifizierungsmethoden, Data Stores und Sitzungseinstellungen zugeordnet.

25 dt. Bereiche

5.3.2 Directory Server

Für den Test des Prototyps war es erforderlich, einen User zu haben, der sich gegenüber dem BLS authentifizieren kann und der über Mandanten- und Bedienernummer verfügt. Die beiden Nummern werden für die Initialisierung einer neuen Session im BLS benötigt. Zu diesem Zweck wurde auf dem Ubuntu Server eine Instanz des frei erhältlichen OpenLDAP installiert. Darin wurde ein Schema mit der Objektklasse `leasmanUser` und als Testperson ein „leasmantester“ angelegt.

Die Schemadatei `/etc/ldap/schema/leasman.schema`:

```
attributetype ( 1.1.1.0.1
                NAME ( 'mdnr' 'MandantenNr' )
                DESC 'Die Leasman Mandantennummer'
                EQUALITY integerMatch
                ORDERING integerOrderingMatch
                SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
                SINGLE-VALUE )

attributetype ( 1.1.1.0.2
                NAME ( 'bdnr' 'BedienerNr' )
                DESC 'Die Leasman Bedienernummer'
                EQUALITY integerMatch
                ORDERING integerOrderingMatch
                SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
                SINGLE-VALUE )

objectclass ( 1.1.1.0
                NAME 'leasmanUser'
                DESC 'repraesentiert einen Leasman User'
                SUP inetOrgPerson
                MUST ( bdnr $ mdnr )
                )
```

Mit Hilfe des Kapitels 7.3 „Eigene Schemas entwerfen“ aus [ldp03] gelang die Definition neuer *AttributeTypes* und *ObjectClasses* in einem LDAP-Schema.

Attributtypen und Objektklassen

Die Wurzeln von LDAP liegen im X.500-Standard, der ein objektorientiertes Datenmodell implementiert. Ein LDAP-Schema beschreibt Attributtypen und Objektklassen. Beide werden als Objekte behandelt. Attributtypen sind Bestandteile von Objektklassen, werden ihnen aber nicht explizit zugeordnet, sodass sie von mehreren Klassen verwendet werden können.

Definitionen von Attributen und Klassen beginnen mit der Angabe eines *Object Identifier* (OID). OIDs sind weltweit einmalig. Sie bestehen aus aufeinander folgenden Ziffern, die durch Punkte getrennt werden. Einige sind für Institutionen reserviert. Die oben angegebenen OIDs sind fiktiv gewählt, da sie nur für diesen speziellen Fall verwendet werden. Weitere gemeinsame Elemente zur Definition sind ein von Menschen lesbarer Name und eine Beschreibung (`DESC`). Der Name kann mehrere, in einfachen Anführungszeichen stehende und durch Leerzeichen getrennte Einträge enthalten. Über den Namen wird das jeweilige Objekt referenziert.

Attributtypen werden durch folgende Elemente weiter definiert:

`EQUALITY`	Angabe zum Test auf Gleichheit; abhängig vom verwendeten „Datentyp“ (siehe Syntax)
`ORDERING`	Angabe einer Regel zur Anordnung
`SYNTAX`	Verwendete Syntax als OID; entspricht dem Datentyp des Attributes; die OID 1.3.6.1.4.1.1466.115.121.1.27 steht für *Integer*
`SINGLE-VALUE`	Gibt an, dass dieses Attribut nur einen Wert enthalten darf; per default sind mehrere Werte erlaubt (`MULTI-VALUE`), was nicht explizit angegeben wird

Objektklassen können Attributdefinitionen von anderen Klassen erben. Das Schlüsselwort `SUP` in der Klassendefinition zeigt an, dass hier von einer Superklasse geerbt wird. Attribute können in Objektklassen entweder optional, durch das Schlüs-

selwort `MAY` gekennzeichnet, oder obligatorisch, zu erkennen am Schlüsselwort `MUST`, eingesetzt werden.

Eine wichtige und in Organisationsstrukturen verbreitete Objektklasse ist `inetOrgPerson`. Die Klasse hat selbst zunächst nur optionale Attribute wie `uid`, `mail`, `homePhone` oder `employeeNumber`. Sie erbt jedoch von der Klasse `organizationalPerson`, die wiederum `person` als Superklasse hat. `person` besitzt unter anderem das obligatorische Attribut `CommonName`, das oft unter dem Alias `cn` bei LDAP-Abfragen verwendet wird, um ein Objekt anzusprechen, das eine Person repräsentiert.

Da die eigens erzeugte Klasse `leasmanUser` ihrerseits von `inetOrgPerson` erbt, kann eine Instanz dieser Klasse bei einer Abfrage durch einen LDAP-Client implizit über `CommonName` oder `cn` angesprochen werden.

Schema einbinden und verwenden

In der Konfigurationsdatei `/etc/ldap/slapd.conf` des Standalone LDAP-Servers wird die Schemadatei eingebunden:

```
# Schema and objectClass definitions
include         /etc/ldap/schema/core.schema
include         /etc/ldap/schema/cosine.schema
include         /etc/ldap/schema/nis.schema
include         /etc/ldap/schema/inetorgperson.schema
# customized schemas
include         /etc/ldap/schema/leasman.schema
```

Um das neu angelegte Schema verwenden zu können, muss der LDAP-Server neu gestartet werden:

```
root@woteba:~#  /etc/init.d/slapd restart
Stopping OpenLDAP: slapd.
Starting OpenLDAP: slapd.
```

Über einen LDAP-Editor kann nun auf Basis der Objektklasse ein Test-User erstellt werden. In diesem Fall geschah das über das ApacheDS[26]-Plugin der Eclipse IDE.

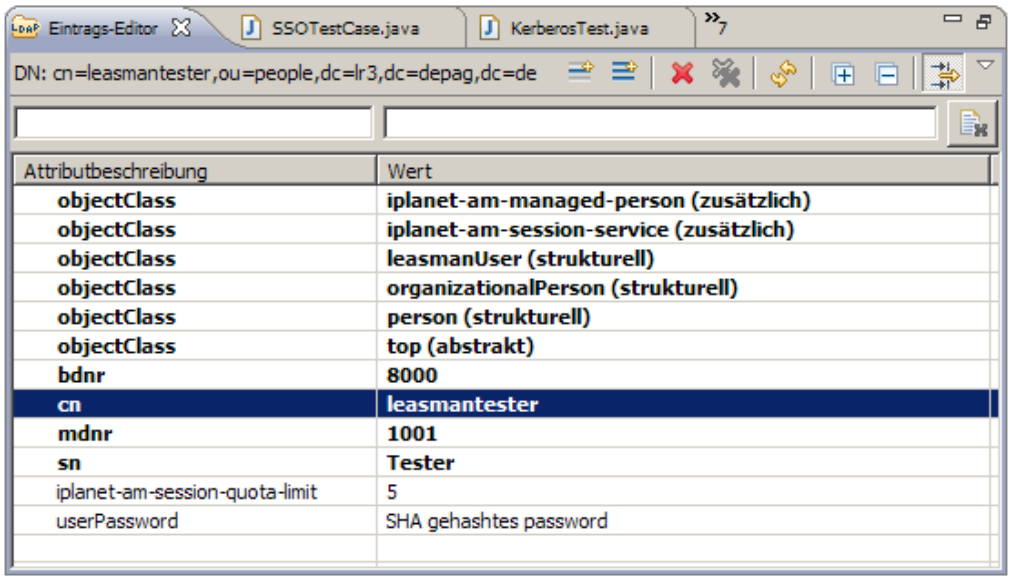


Abb. 5-7: Eclipse LDAP-Editor

5.3.3 Data Store registrieren

Der standardmäßig verwendete Nutzerdatenspeicher des *Top Level Realm* von OpenSSO ist „embedded“ und meint den LDAPv3-konformen *Open Directory Service* (OpenDS), welcher bei der Installation automatisch eingerichtet wird. Um den im OpenLDAP angelegten User hier zu verwenden, also ihm die Möglichkeit zu geben, sich am OpenSSO-Server anzumelden und SSOTokens benutzen zu können, muss eine OpenLDAP-Instanz als *Data Store* hinzugefügt werden. Dabei sind unter anderem Angaben zum Server und Port sowie zum LDAP-Administrator notwendig. In der Administrationskonsole erscheint dann unter *Access Control* > *Subjects* der Nutzer „leasmantester“.

26 Apache Directory Studio: siehe http://directory.apache.org/

5.4 Implementierung für JOSSO

Zur Demonstration der Austauschbarkeit des SSOService wurde das entworfene Interface für ein weiteres System exemplarisch implementiert. Diese Implementierung des SSOService für Java Open Single Sign-On (JOSSO) ist vom Umfang her kaum mit der OpenSSO-Implementierung zu vergleichen.

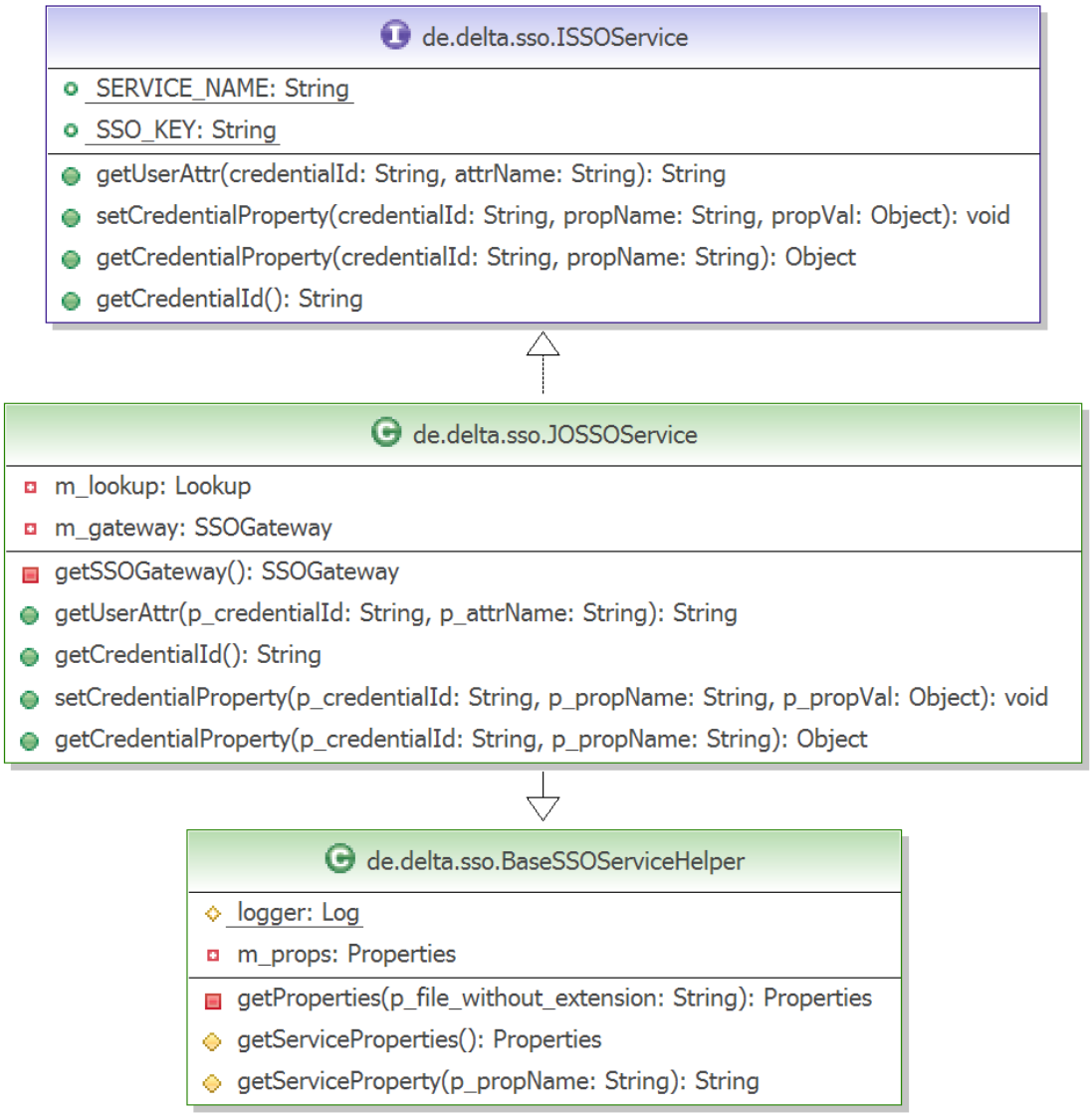


Abb. 5-8: JOSSOService UML

5.4.1 SSO-Gateway

Während der OpenSSOService über das zur Verfügung gestellte Client SDK auf den SSO-Server zugreift und dabei nur einen geringen Teil des komplexen Funktionsumfanges nutzt, wird bei der Implementierung für JOSSO eine Art *Standalone SSO Gateway* zur Verfügung gestellt.

Es wird kein laufender Application Server benötigt. Der JOSSOService startet zur Laufzeit selbst eine eigene Gateway-Instanz, was einem SSO-Server bei JOSSO entspricht. Der Gateway stellt alle benötigten Komponenten zur Laufzeit zur Verfügung und wird nach Ablauf eines Unit-Tests beziehungsweise nach dem Schließen der Anwendung mit beendet.

Dieses SSO-Testsystem ist nicht persistent, so dass alle aktiven Sessions, beziehungsweise die sogenannten Assertions, im Hauptspeicher gehalten werden und nach gelaufenem Test oder Anwendungsende verloren gehen. Daher ist es nicht möglich, beim Start auf vorhandene gültige Sessions beziehungsweise Assertions zu testen. Im Produktiveinsatz ist der JOSSO-Gateway auf einem Application Server installiert und kann Komponenten wie das Session-Management als vollwertige Dienste zur Verfügung stellen.

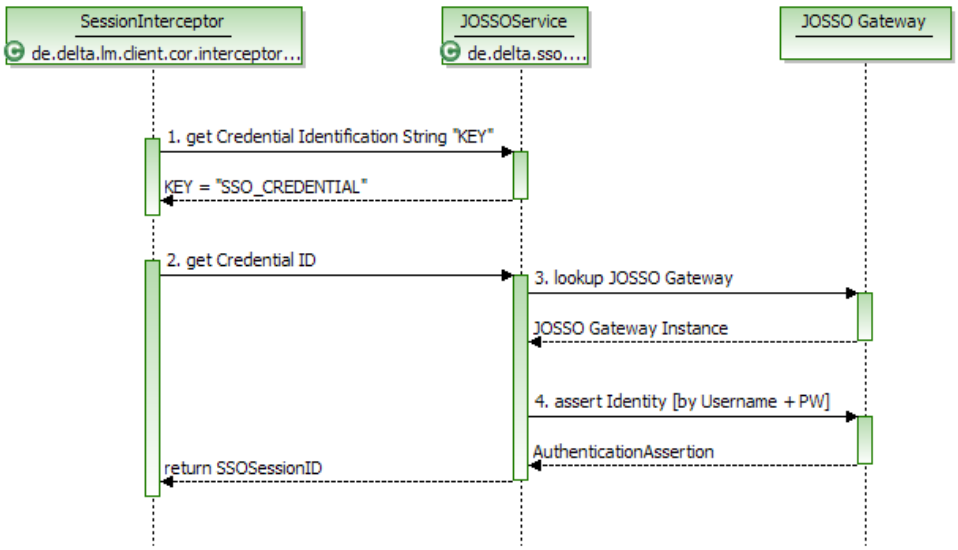


Abb. 5-9: JOSSOService clientseitig

5.4.2 User Attributes

Die Methode des SSOService zum Auslesen von nutzerbezogenen Attributen, beispielsweise aus einem LDAP-Verzeichnis, wird bei der JOSSO-Implementierung über die Funktionen `findUserInSession()` des Gateway sowie `getProperties()` des darüber erhaltenen SSOUser realisiert. Man erhält die Properties in Form eines `SSONameValuePair`, also eine Collection mit Namen und Werten.

Zum Testen des JOSSOService wurde dieselbe LDAP-Instanz wie für OpenSSO verwendet. Die Konfiguration derselben wird im ersten Teil des Abschnitts Directory Server unter 5.3.2 beschrieben.

Um die im SSOInterceptor benötigte Mandanten- und Bedienernummer aus dem OpenLDAP durch das oben genannte SSONameValuePair erhalten zu können, müssen in der Konfigurationsdatei `josso-gateway-ldap-stores.xml` in der Bean mit der ID „josso-identity-store“ für `userPropertiesQueryString` die Einträge „cn=cn,mdnr=mdnr,bdnr=bdnr“ vorhanden sein:

```
<ldap-istore:ldap-bind-store
 id="josso-identity-store"
 initialContextFactory="com.sun.jndi.ldap.LdapCtxFactory"
 providerUrl="ldap://10.1.10.111:389"
 securityPrincipal="cn=admin,dc=lr3,dc=depag,dc=de"
 securityCredential="ldapadminpwd"
 securityAuthentication="simple"
 ldapSearchScope="SUBTREE"
 usersCtxDN="ou=people,dc=lr3,dc=depag,dc=de"
 principalUidAttributeID="cn"
 uidAttributeID="uniquemember"
 updateableCredentialAttribute="userPassword"
 userPropertiesQueryString="cn=cn,mdnr=mdnr,bdnr=bdnr"
/>
```

Dabei ist das Format der kommaseparierten Attribute:

```
LDAP_Attribut_Name = User_Attribut_Name.
```

Als LDAP-Attribut muss ein in der LDAP-Instanz Vorhandenes angegeben werden. Kann dieses nicht gefunden werden, wird beim Starten des Gateway eine entsprechende Warnung ausgegeben. Das User-Attribut ist beliebig zu wählen; hier wurde der Einfachheit halber der Name des LDAP-Attributes übernommen. Der Bezeichner muss aber auf jeden Fall demjenigen im SSOInterceptor bei `createSession()` entsprechen:

```
sid = m_sessionService.createSession(
 Long.parseLong(m_ssoService.getUserAttr( credentialId,"mdnr" )),
 Long.parseLong(m_ssoService.getUserAttr( credentialId,"bdnr" ))
);
```

5.4.3 Credential Properties

Die im Sinne des SSOService als Credentials verwendeten AuthenticationAssertions bieten ebensowenig wie die Klasse SSOSession aus der JOSSO-API eine Möglichkeit, benutzerdefinierte Eigenschaften an die Instanzen zu hängen, so wie es bei OpenSSO die SSOTokens mit `getProperty()` und `setProperty()` tun.

Durch die Notwendigkeit der Speicherung der BLS Session-ID im Credential-Objekt wurde hier eine Erweiterung der bestehenden Klasse `AuthenticationAssertion` erforderlich. Diese Erweiterung ist in `org.josso.gateway.assertion.ExtendedAuthenticationAssertion` implementiert. Die Klasse bietet Methoden zum Setzen und Auslesen von Properties und wird im JOSSOService anstelle der von JOSSO gelieferten Basisklasse verwendet.

5.5 Test

Um die Integration der Geschäftsanwendung in SSO-Systeme durch den SSOService zu testen, wurde ein einfacher JUnit-Test namens `SSOTestCase` verwendet, der auf einem vorhandenen `LocalBlsBasicTestCase` aufbaut. Im Test wird vom PartnerService des BLS eine Collection von Anredeformen über `getAllSalutations()` abgefragt. Der Inhalt der Rückgabe ist nicht relevant für den Erfolg des Testfalles. Wichtig ist nur, dass dieser erfolgreich durchlaufen werden kann.

Über die Ausgabe in der Konsole kann der Ablauf nachvollzogen werden. Für die JOSSO-Implementierung sieht das Ergebnis wie folgt aus. Die Formatierung wurde zugunsten der Lesbarkeit angepasst. Es werden einerseits Informationen angezeigt, die aus Logs des JOSSO-Systems stammen. Die Basisklasse der entwickelten SSO-Service-Implementierungen `BaseSSOServiceHelper` stellt ebenfalls Logging-Fähigkeiten zur Verfügung. Die entsprechenden Zeilen sind durch das vorangestellte „`[ISSOService]`" zu erkennen. Hinzu kommen Ausgaben, die im Zusammenhang mit dem Testfall von zugrundeliegenden BLS-Komponenten und dem Spring-Framework geliefert werden.

Nach einer langen Liste aus Bean-Definitionen, dem Laden von `properties`-Dateien und weiteren Initialisierungen, folgen die für den Test eigentlich relevanten Informationen. Zunächst wird JOSSO anhand der XML-Konfigurationsdateien eingerichtet:

```
INFO  [Lookup] Init resourceName </org/josso/gateway/conf/josso-gateway-config.xml>
INFO  [SecurityDomainRegistryImpl] Registering SecurityDomain : josso
INFO  [SSOSessionManagerImpl] [initialize()] : MaxInactive.................=30
INFO  [SSOSessionManagerImpl] [initialize()] : MaxSessionsPerUser..........=-1
INFO  [SSOSessionManagerImpl] [initialize()] : InvalidateExceedingSessions.=false
INFO  [SSOSessionManagerImpl] [initialize()] : SesisonMonitorInteval.......=10000
INFO  [SSOSessionManagerImpl] [initialize()] : Restore Sec.Domain Registry.=josso
```

Der SSOService lädt den Test-User „leasmantester“ aus der entsprechenden `properties`-Datei. JOSSO erzeugt für diesen eine neue SSO-Session mit zugehöriger ID unter Verwendung des „basic“-Authentifizierungsschemas:

```
INFO  [ISSOService] SSOService.properties loaded successfully
INFO  [AUDIT] Wed Dec 09 08:10:40 CET 2009 - sso-session - info - leasmantester -
      createSession=success - ssoSessionId=2841B95197A9E3BC2CBC77BAEAE45D68
INFO  [AUDIT] Wed Dec 09 08:10:40 CET 2009 - sso-user - info - leasmantester -
      authenticationSuccess=success - authScheme=basic-authentication,
      ssoSessionId=2841B95197A9E3BC2CBC77BAEAE45D68
```

Anhand der von JOSSO erzeugten SSO Session-ID wird versucht, aus dem Credential die BLS Session-ID auszulesen. Diese ist hier `null`, was eine Fehlermeldung hervorruft, die besagt, dass keine SID gefunden wurde und eine neue Session erzeugt wird. Dazu werden wiederum die beiden Attribute `mdnr` und `bdnr` benötigt, die aus dem LDAP-Kontext von leasmantester bezogen werden.

```
DEBUG [ISSOService] GET ID = 2841B95197A9E3BC2CBC77BAEAE45D68
DEBUG [ISSOService] 2841B95197A9E3BC2CBC77BAEAE45D68: GET SID = null
ERROR [SSOInterceptor] SID not found in Credential > createSession() > getSessionID
DEBUG [ISSOService] 2841B95197A9E3BC2CBC77BAEAE45D68: GET mdnr = 1001
DEBUG [ISSOService] 2841B95197A9E3BC2CBC77BAEAE45D68: GET bdnr = 8000
DEBUG [ISSOService] 2841B95197A9E3BC2CBC77BAEAE45D68:
      SET SID = 9bfd468e-9ad0-4554-9d7b-08a4794002a8
```

Es folgen BLS-Logs, welche die Verwendung der oben genannten Attribute sowie die erzeugte Session-ID bestätigen. Als Abschluss meldet der `SSOTestCase`, dass er erfolgreich durchlaufen wurde.

```
DEBUG [MandantDataSourceInterceptor] Using DataSource of mandant 1001
DEBUG [AbstractStoredProcedure] PROCEDURE CALL dpbdnr_register
      (in.bdnr='8000', in.bsid='9bfd468e-9ad0-4554-9d7b-08a4794002a8',
      in.del_inactives='T',in.sid='687406')
DEBUG [SessionManager]
      recoverSession [9bfd468e-9ad0-4554-9d7b-08a4794002a8:1001:8000]
# SSOTestCase: [SUCCESS]
```

In Eclipse wird der fehlerlose Test durch einen grünen Haken in der JUnit-View angezeigt. Zudem stehen „Errors“ und „Failures“ auf 0.

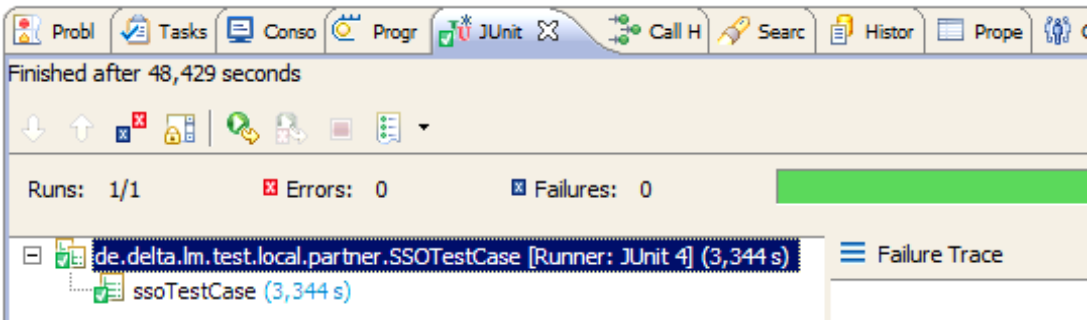


Abb. 5-10: Erfolgreicher JUnit-Test in Eclipse

5.6 Probleme

„Die Anforderungen der Benutzer sind vage, weil sie noch nie ein solches System gesehen haben. Sie stehen einer Menge Unsicherheiten gegenüber, weil Sie unbekannte Algorithmen, Techniken und Programmiersprachen verwenden. Projekte brauchen Zeit, Sie können sicher sein, dass sich Ihre Arbeitsumgebung verändern wird, noch bevor Sie fertig sind."
[prg03] S. 44

Ungültige SSOTokenIDs bei OpenSSO

Bei der Implementierung für OpenSSO wird eine Credential-ID (CID) als Header-Parameter des CommandTO (CTO) angehängt. Die CID gehört zu einem SSOToken, welches bei jedem Aufruf neu erzeugt wird. Serverseitig wird im SSOInterceptor aus der im CTO enthaltenen CID das Token wieder hergestellt, um Informationen aus diesem zu lesen.

Durch dieses Vorgehen wird bei jedem Start des vorliegenden Testfalles also ein neues Token erzeugt, das nur einmal wieder verwendet wird und dazu beiträgt, dass die Anzahl der Token im Repository auf dem SSO-Server ständig inkrementiert wird.

In einer früheren Variante der Implementierung wurde anstelle der permanenten Neuerzeugung von Tokens eines aus der Liste der aktuell gültigen SSO-Sessions via `SSOTokenManager.getValidSessions()` geholt, sofern für den zu authentifizierenden Nutzer eine laufende Session existierte. Solange man das gefundene Token direkt als Objekt an ein CTO hängt und später dessen gespeicherte Nutzerinformationen wie die BLS Session-ID abfragt, gibt es keine Probleme, falls es dann noch gültig sein sollte. Soll jedoch jeweils im Zuge der Verwendung dieses Tokens ein Refresh der zugehörigen Session gemacht werden, scheitert dieses an einer Ausnahme, die besagt, dass die Session-ID – gemeint ist die SSOTokenID – ungültig sei. Daher kann dieses Token zwar, im Gegensatz zur oben genannten späteren Variante, bei einer gewissen Anzahl von Tests wiederverwendet werden, allerdings nur so lange, bis die *Idle Time* (Leerlaufzeit) abgelaufen ist. Folglich wird nie die maximale Gültig-

keitszeit einer SSO-Session erreicht werden können. Zu Testzwecken kann man zwar die Idle Time sehr hoch einstellen, doch würde das im produktiven Einsatz eine prinzipielle Zweckentfremdung bedeuten.

Bei der Untersuchung des aufgetretenen Fehlers wurde klar, dass die angeblich ungültige SSOTokenID nur ein sogenanntes *Session Handle* war. Das Token ließ sich nicht anhand seiner ID wiederherstellen.

Eine SSOTokenID ist eine Zeichenkette, die sich wie folgt zusammensetzt[27]: `<opaque core session prefix>@<session extension fields>#<http session id>.` Das Session Handle ist scheinbar genau dieser String, nur mit dem Präfix `shandle` . Der Versuch, diesen Präfix durch eine Zeichenkettenfunktion abzuschneiden, brachte keinen Erfolg. Trotz intensiver und zeitaufwendiger Suche in Dokumentationen, Beispielen und Entwicklerkommentaren sowie der testweisen Verwendung von Klassen und Schnittstellen aus dem Session-Kontext von OpenSSO, gelang es nicht, aus dem erhaltenen Session Handle wieder ein vollwertiges SSOToken zu reproduzieren.

In der API Dokumentation von OpenSSO ist zur Funktion `getValidSessions()` der Klasse SSOTokenManager zu lesen:

> *"Returns a list of single sign on token objects which correspond to valid Sessions accessible to requester. Single sign on tokens returned are restricted: they can only be used to retrieve properties and destroy sessions they represent." [api]*

Die hier genannten Einschränkungen trafen beim Test allerdings nur teilweise zu. Es war möglich, gespeicherte Properties der Tokens auszulesen. Das Terminieren der Session gelang jedoch nicht. Auch hier wurde derselbe Fehler erzeugt wie beim oben genannten Refresh-Versuch. Die Idee dabei war, ein gültiges Token zu nutzen und dieses aufgrund der „ungültigen" Session-ID nicht direkt zu verwenden sondern die gespeicherten Informationen in ein neues Token zu übertragen und anschließend das Alte zu verwerfen.

27 vgl. [oss] S. 29

Die mehrfache Wiederverwendung eines Tokens kommt zwar dem eigentlichen Gedanken des Single Sign-On schon näher, doch konnte diese aus den genannten Gründen nicht verwirklicht werden. Zudem erlaubte die Forderung nach einer hohen Abstraktionsstufe der Schnittstelle keine Objekte vom Typ SSOToken sondern verlangte nach einer Zeichenkette zur Identifikation solcher Credential-Objekte. Auch daher musste von der genannten Variante abgesehen werden.

6 Fazit

6.1 Auswertung

Das **Ziel** der Diplomarbeit war, einen Software-Prototypen zu erstellen, der eine Integration des Leasman Business Logic Servers in Single Sign-On Systeme ermöglicht. Vorhandene SSO-Systeme aus dem OpenSource-Bereich, die auf Java basieren, wurden untersucht. Um die Geschäftsanwendung in verschiedene Systeme integrieren zu können, wurde aus den ermittelten Gemeinsamkeiten eine Schnittstelle konzipiert. Sie konnte für zwei SSO-Lösungen implementiert und erfolgreich getestet werden.

Die existierenden SSO-Lösungen besitzen sehr unterschiedliche Herangehensweisen. Es besteht keine Standardlösung für die Umsetzung von Single Sign-On. Die Systeme basieren jedoch auf Standards wie JAAS zur Authentisierung, LDAP für den Zugriff auf Datenspeicher und Frameworks wie Spring und EJB.

Single Sign-On wird teilweise unterschiedlich interpretiert. Zum einen wird es als Möglichkeit beschrieben, nach der Authentisierung an einer Anwendung mit Hilfe von Tickets weitere Anwendungen in der SSO-Umgebung ohne Re-Authentisierung nutzen zu können. Andererseits wird mit SSO auch die Vorgehensweise bezeichnet, nach dem ersten Anmeldevorgang über eine lange Zeit, auch über den Neustart des zugrundeliegenden Systems hinweg, angemeldet zu bleiben. Eine dritte Interpretation kombiniert beide Ansätze.

OpenSSO war zunächst das einzige System, mit dem eine Kommunikation zwischen Test Client und SSO-Server überhaupt möglich wurde. Mit JOSSO gelang das durch Verwendung der API auch grundlegend, wodurch es auch weiterhin als Kandidat mit berücksichtigt wurde.

Für die Tests kamen verschiedene Serverkomponenten zum Einsatz. Unter anderem wurden Application Server in den zum Zeitpunkt der Bearbeitung aktuell verfügbaren Versionen verwendet. Des Weiteren konnten Erfahrungen mit LDAP-basierten Datenspeichern gesammelt werden.

Bisher nicht vorhanden und daher empfehlenswert sind Testszenarien, bei denen sich Leasman und andere Clients am BLS anmelden und durch die gemeinsame Nutzung von Tickets Single Sign-On im eigentlichen Sinne demonstrieren.

Als **Ergebnis** entstand ein Java Interface mit exemplarischen Referenzimplementierungen für die Single Sign-On Systeme OpenSSO und JOSSO. Mit den entwickelten Implementierungen wurde die Realisierbarkeit der Integration nachgewiesen. Damit konnte dieser Teil der Anforderungen erfüllt werden. Authentisierung und Identitätsmanagement erfolgen über das SSO-System, das seinerseits wichtige Sicherheitsaspekte implementiert.

Als sehr hilfreich für die Bearbeitung erwies sich die Versionsverwaltung. Ausgehend von ersten Erfahrungen bei der Verwendung der BLS-Sourcen via CVS (Concurrent Versions System) und SVN (Subversion), wurden die Versionen des selbst geschriebenen Quellcode ebenso verwaltet wie das vorliegende Dokument inklusive aller damit im Zusammenhang stehenden Daten. Das Subversion Projekt der Hochschule Mittweida stellt freundlicherweise einen passenden Server zu diesem Zweck zur Verfügung.

Die Einarbeitung in das Thema Single Sign-On brachte besondere Herausforderungen mit sich, da eine große Bandbreite an Technologien betrachtet werden muss. Dazu gehören Authentisierung und Identitätsmanagement mit der verschlüsselten Übertragung von Nutzerdaten. Aus der breiten Palette mussten Komponenten herausgefiltert werden, die zur Integration der Geschäftsanwendung notwendig waren.

Mit dem Ergebnis ist ein wichtiger Schritt zur Reduzierung der interaktiven Anmeldevorgänge erfolgt.

6.2 Ausblick

Beim erstellten Prototyp steht die Authentisierung im Vordergrund. Falls man sich langfristig für SSO entscheidet, kann man neben **Authentication**- auch **Authorization**-, **Identity**- und **Sessionmanagement** an ein solches System delegieren. OpenSSO bietet umfassende Konfigurationsmöglichkeiten zu diesen Komponenten. Die Integrationsschnittstelle müsste dahingehend erweitert werden. Zudem bringt OpenSSO Clusterfähigkeit mit, sodass im Falle der oben genannten Erweiterung die Verteilung von Ressourcen und damit die Performance optimiert werden könnte.

Falls weiterhin nur die Authentisierungskomponente von SSO durch den BLS genutzt werden soll, wäre ein produktiver Einsatz von JOSSO denkbar. Es ist im Gegensatz zu OpenSSO ein leichtgewichtiges System. Es müsste ein SSO-Gateway installiert werden, was mit Hilfe der Auslieferung des Tomcat inklusive bereits deploytem JOSSO hinreichend einfach möglich wäre.

Ein Fall, der bisher außer Acht gelassen wurde, ist die Änderung von Mandanten- oder Bedienernummer. Bei der Anmeldung am Leasman Client kann nach Eingabe von Bediener und Kennwort ein Mandant gewählt werden. Diese Auswahl wird durch den vorliegenden Prototyp nicht abgebildet.

Im Hinblick auf die Wiederverwendung bestehender SSOTokens bei OpenSSO ist weiterhin zu prüfen, ob es für die in Kapitel 5.6 beschriebenen Probleme Lösungsansätze seitens der Entwickler oder Anwender gibt.

Sicherheitsaspekte wurden bei der Umsetzung grundlegend beachtet, standen aber für den Prototypen nicht im Vordergrund. Dieser Punkt muss bei einer Weiterverfolgung der Thematik unbedingt berücksichtigt werden. Ergänzend ist zu sagen, dass parallel zur Bearbeitung des vorliegenden Themas Sicherungsmechanismen für die Kommunikation mit dem Business Logic Server implementiert werden, was

auch einen Schutz der mit übertragenen SSO Session-ID zur Folge hat. Resultierend wird so auch ein unautorisierter Zugriff auf Nutzerdaten erschwert.

Die Implementierung der entworfenen Schnittstelle ist mit vertretbarem Aufwand austauschbar. Eine Untersuchung weiterer SSO-Lösungen wird zeigen, inwieweit Anpassungen oder Erweiterungen notwendig werden. Die prototypische Umsetzung könnte daran anknüpfend in eine Version für den produktiven Einsatz überführt werden.

Anhang

A SSOService Schnittstelle

```
/**
 * Der <code>SSOService</code> stellt abstrakte Methoden zur Integration
 * in Single Sign-On Systeme bereit.
 *
 * @author pemmann
 * @version 2009.10.05
 */

package de.delta.sso;

public interface ISSOService
{
  public static final String SERVICE_NAME = "SSOService";
  public static String SSO_KEY = "SSO_CREDENTIAL_ID";

  /**
   * @return Zeichenkette zur Identifikation eines Credential
   * (= Nachweis zur Benutzerauthentisierung)
   */
  public String getCredentialId();

  /**
   * @param credentialId  Zeichenkette zur Credential Identifikation
   * @param propName    Der Name der Eigenschaft des Credential, dessen Wert
   * ermittelt werden soll
   * @return Der Wert der geforderten Credential-Eigenschaft als Objekt
   */
  public Object getCredentialProperty(String credentialId, String propName);

  /**
   * @param credentialId  Zeichenkette zur Credential Identifikation
   * @param propName    Der Name der Eigenschaft des Credential, dessen Wert
   * gesetzt werden soll
   * @param propVal    Der Eigenschaftswert
   */
  public void setCredentialProperty(String credentialId, String propName,
   Object propVal);

  /**
   * @param credentialId  Zeichenkette zur Credential Identifikation
   * @param attrName    Bezeichner des Attributes aus dem LDAP-Kontext des
   * Benutzers
   * @return Wert des LDAP-Attributes
   */
  public String getUserAttr(String credentialId, String attrName);

}
```

B SSOService Helferklasse

```
/**
 * Die Basisklasse fuer SSOService-Implementierungen
 * stellt Hilfsfunktionen zur Verfuegung
 *
 * @author pemmann
 * @version 2009.11.11 (project: SSOService)
 */

package de.delta.sso;

import java.util.Properties;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class BaseSSOServiceHelper
{
  protected static final Log logger = LogFactory.getLog(ISSOService.class);
  private Properties m_props = null;

  /**
   * @param p_file_without_extension  "Properties"-Datei ohne Dateiendung
   angeben
   * @return {@link Properties} oder <code>null</code> wenn Datei nicht
   gefunden
   */
  private Properties getProperties(String p_file_without_extension)
  {
    Properties props = new Properties();
    String propext = ".properties";
    if(!p_file_without_extension.contains(propext)) p_file_without_extension
   += propext;

    try {
      props.load(ClassLoader.getSystemResourceAsStream(
     p_file_without_extension));
      logger.info(p_file_without_extension + " loaded successfully");
      return props;
    } catch (Exception e) {
      logger.error(p_file_without_extension + " not found");
      return null;
    }
  }

  /**
   * @return {@link java.util.Properties}
   */
  protected final Properties getServiceProperties()
  {
    if (m_props == null) m_props = getProperties(ISSOService.SERVICE_NAME);
    return m_props;
  }

  /**
   * @param p_propName Property aus ServiceProperties
   * @return Wert der Property
   */
  protected final String getServiceProperty(String p_propName)
  {
    return getServiceProperties().getProperty(p_propName);
  }

}
```

C Implementierung für OpenSSO

```
/**
 * Implementierung der <code>SSOService</code> Schnittstelle fuer OpenSSO
 *
 * @author pemmann
 * @version 2009.11.10 (project: OpenSSO-3)
 */

package de.delta.sso;

import java.security.AccessController;
import java.util.Iterator;

import javax.security.auth.callback.Callback;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.callback.UnsupportedCallbackException;
import javax.security.auth.login.LoginException;

import com.iplanet.am.util.SystemProperties;
import com.iplanet.sso.SSOException;
import com.iplanet.sso.SSOToken;
import com.iplanet.sso.SSOTokenManager;
import com.sun.identity.authentication.AuthContext;
import com.sun.identity.idm.AMIdentity;
import com.sun.identity.idm.IdRepoException;
import com.sun.identity.idm.IdUtils;
import com.sun.identity.security.AdminTokenAction;
import com.sun.identity.shared.encode.Base64;

public class OpenSSOService extends BaseSSOServiceHelper implements ISSOService
{
   private SSOTokenManager m_ssoManager = null;

   /**
    * @throws {@link com.iplanet.sso.SSOException}
    * @return {@link com.iplanet.sso.SSOTokenManager} Instanz
    */
   private final SSOTokenManager getSSOTokenManager() throws SSOException
   {
      if (m_ssoManager == null) {
         m_ssoManager = SSOTokenManager.getInstance();
      }
      return m_ssoManager;
   }

   public String getCredentialId()
   {
      SSOToken _token = null;

      try {

         if(Boolean.valueOf(getServiceProperties()
               .getProperty("getvalidsessions"))) {
            _token = getTokenFromValidSessions();

         } else {

            _token = getSessionToken(
                  getServiceProperties().getProperty("user.uid"),
                  new String(Base64.decode(getServiceProperties()
                      .getProperty("user.pass"))));
         }
```

```
    } catch (SSOException e) {
       logger.error(e);
    } catch (Exception e) {
       logger.error(e);
    }
    return _token.getTokenID().toString();
}

public Object getCredentialProperty(String p_credentialId,
    String p_propName)
{
    logger.info("Get Property By Credential ID");
    Object propValue = null;

    try {
       SSOToken _token = getSSOTokenById(p_credentialId);
       logger.debug("token.getPrincipal(): " +
          _token.getPrincipal().getName());
       propValue = _token.getProperty(p_propName);

    } catch (SSOException e) {
       logger.error("getCredentialProperty(): " + e);
    }
    return propValue;
}

public void setCredentialProperty(String p_credentialId, String p_propName,
    Object p_propVal)
{
    try {
       getSSOTokenById(p_credentialId).setProperty(p_propName,
          (String)p_propVal);
    } catch (SSOException e) {
       logger.error("setCredentialProperty(): " + e);
    }
}

public String getUserAttr(String p_credentialId, String p_attrName)
{
    try {
       AMIdentity ami = IdUtils.getIdentity(getSSOTokenById(p_credentialId));
       return ami.getAttribute(p_attrName).iterator().next().toString();

    } catch (SSOException e) {
       logger.error(e);
    } catch (IdRepoException e) {
       logger.error(e);
    }
    return null;
}

/**
 * Ueber den {@link com.iplanet.sso.SSOTokenManager}
   ein {@link com.iplanet.sso.SSOToken} aus einer TokenID erzeugen
 *
 * @param p_tokenId     Zeichenkette zur Identifikation des SSOToken
 * @return {@link com.iplanet.sso.SSOToken}
 */
private SSOToken getSSOTokenById(String p_tokenId)
{
    SSOToken _token = null;

    try {
       _token = getSSOTokenManager().createSSOToken(p_tokenId);
       getSSOTokenManager().validateToken(_token);
```

```
    } catch (SSOException e) {
       logger.error(e);
    }
    return _token;
}

/**
 * neue SSO Session mit Token fuer angebenen User erstellen
 * @param p_userId      Benutzeridentifikation in Form eines eindeutigen
                        LDAP DN (Distinguished Name)
 * @param p_passwd      Benutzerpasswort
 * @see TokenUtils
 */
private SSOToken getSessionToken(String p_userId, String p_passwd)
throws Exception
{
   AuthContext ac;

   try {
      ac = new AuthContext(SystemProperties
         .get("com.iplanet.am.services.deploymentDescriptor"));
      ac.login();

   } catch (LoginException le) {
      logger.error(le);
      logger.error("Failed to create new AuthContext: Naming Service is not
         available.");
      return null;
   }

   try {
      Callback[] callbacks = null;
      // Get the information requested by the plug-ins
      if (ac.hasMoreRequirements())
      {
         callbacks = ac.getRequirements();

         if (callbacks != null)
         {
            addLoginCallbackMessage(callbacks, p_userId, p_passwd);
            ac.submitRequirements(callbacks);

            if (ac.getStatus() == AuthContext.Status.SUCCESS)
            {
               logger.info(p_userId + ": Auth success");
            } else if (ac.getStatus() == AuthContext.Status.FAILED) {
               logger.error(p_userId + ": Authentication has FAILED");
            }
         }
      }
   } catch (UnsupportedCallbackException uce) {
      logger.error("UnsupportedCallbackException!");
      return null;
   }

   return ac.getSSOToken();
}

/**
 * @param p_callbacks
 * @param p_userId
 * @param p_passwd
 * @throws UnsupportedCallbackException
 */
private void addLoginCallbackMessage(Callback[] p_callbacks,
   String p_userId, String p_passwd) throws UnsupportedCallbackException
```

```
    {
       int i = 0;
       try {
          for (i = 0; i < p_callbacks.length; i++) {
             if (p_callbacks[i] instanceof NameCallback) {
                NameCallback nc = (NameCallback) p_callbacks[i];
                nc.setName(p_userId);
             } else if (p_callbacks[i] instanceof PasswordCallback) {
                PasswordCallback pc = (PasswordCallback) p_callbacks[i];
                pc.setPassword(p_passwd.toCharArray());
             }
          }
       } catch (Exception e) {
          throw new UnsupportedCallbackException(p_callbacks[i],
                "Callback exception: " + e);
       }
    }

    private SSOToken getTokenFromValidSessions() throws Exception
    {
       SSOToken adminToken = (SSOToken) AccessController
          .doPrivileged( AdminTokenAction.getInstance() );
       SSOTokenManager tokenman = SSOTokenManager.getInstance();

       // gueltige Sessions koennen nur mit Admin-Rechten geholt werden
       // aus DOC zu "getValidSessions":
       /*
        * Returns a list of single sign on token objects
        * which correspond to valid Sessions accessible to requester.
        * Single sign on tokens returned are restricted: they can only be used
        * to retrieve properties and destroy sessions they represent.
        */
       Iterator it = tokenman.getValidSessions( adminToken,
            SystemProperties.get("com.iplanet.am.server.host")).iterator();

       SSOToken stok = null;

       while (it.hasNext())
       {
          stok = (SSOToken) it.next();
          if (stok.getPrincipal().toString()
                .equals(getServiceProperties().get("user.uid")))
          {
             logger.debug("[SUCCESS] user '" + getServiceProperties()
                .get("user.uid") + "' has token");

             if (!tokenman.isValidToken(stok))
             {
                logger.error("Token is not valid!");
                continue;
             }
             return stok;
          }
       }
       return null;
    }

}
```

D Implementierung für JOSSO

```
/**
 * Implementierung der <code>SSOService</code> Schnittstelle fuer JOSSO
 *
 * @author pemmann
 * @version 2009.11.11 (project: JOSSO-3)
 */

package de.delta.sso;

import org.josso.Lookup;
import org.josso.auth.Credential;
import org.josso.auth.exceptions.SSOAuthenticationException;
import org.josso.auth.scheme.PasswordCredential;
import org.josso.auth.scheme.UsernameCredential;
import org.josso.gateway.SSOException;
import org.josso.gateway.SSOGateway;
import org.josso.gateway.SSONameValuePair;
import org.josso.gateway.assertion.AuthenticationAssertion;
import org.josso.gateway.assertion.ExtendedAuthenticationAssertion;
import org.josso.gateway.identity.SSOUser;

import com.sun.org.apache.xml.internal.security.exceptions
   .Base64DecodingException;
import com.sun.org.apache.xml.internal.security.utils.Base64;

public class JOSSOService extends BaseSSOServiceHelper implements ISSOService
{
  private Lookup m_lookup = Lookup.getInstance();
  private SSOGateway m_gateway = null;

  /**
    * @return SSOGateway Instanz
    */
  private SSOGateway getSSOGateway()
  {
    try {
      if (m_gateway == null) {
        m_lookup.init("/org/josso/gateway/conf/josso-gateway-config.xml");
        m_gateway = m_lookup.lookupSSOGateway();
        m_gateway.prepareDefaultSSOContext();
      }
    } catch (Exception e) {
      logger.error(e);
    }
    return m_gateway;
  }

  public String getCredentialId()
  {
    SSOGateway gw = getSSOGateway();
    AuthenticationAssertion assertion = null;
    String _id = null;
    Credential[] creds = null;

    try  {
      creds = new Credential[] {
        new UsernameCredential( getServiceProperty("user.uid") ),
        new PasswordCredential( new String(Base64.decode(
     getServiceProperty("user.pass"))) )
      };

      assertion = gw.assertIdentity(creds, "basic-authentication");
```

```
      _id = assertion.getSSOSessionId();
    } catch (SSOAuthenticationException e) {
      logger.error(e);
    } catch (Base64DecodingException e) {
      logger.error(e);
    } catch (SSOException e) {
      logger.error(e);
    }
    return _id;
  }

  public void setCredentialProperty(String p_credentialId, String p_propName,
      Object p_propVal)
  {
    SSOGateway gw = getSSOGateway();
    ExtendedAuthenticationAssertion extauth = null;
    try {
      extauth = new ExtendedAuthenticationAssertion( gw.assertIdentity(
     p_credentialId ).getId() );
      extauth.setProperty(p_propName, (String) p_propVal);
    } catch (SSOException e) {
      logger.error(e);
    }
  }

  public Object getCredentialProperty(String p_credentialId,
   String p_propName)
  {
    SSOGateway gw = getSSOGateway();
    ExtendedAuthenticationAssertion extauth = null;
    Object _value = null;

    try {
      extauth = new ExtendedAuthenticationAssertion( gw.assertIdentity(
     p_credentialId ).getId() );
      _value = extauth.getProperty(p_propName);
    } catch (Exception e) {
      if (_value != null) {
        logger.error(e);
      }
    }

    return _value;
  }

  public String getUserAttr(String p_credentialId, String p_attrName)
  {
    SSOGateway gw = getSSOGateway();
    SSOUser user = null;
    String _value = null;

    try {
        user = gw.findUserInSession(p_credentialId);

         for (SSONameValuePair nvp : user.getProperties()) {
           if (nvp.getName().equals( p_attrName )) {
             _value = nvp.getValue();
           }
      }
    } catch (SSOException e) {
      logger.error(e);
    }
    return _value;
  }
}
```

E Erweitertes Credential bei JOSSO

```
/**
 * Default Authentication Assertion implementation
 *
 * @author pemmann
 * @version $Id: ExtendedAuthenticationAssertion.java 2009-11-06 14:14:00Z
 */

package org.josso.gateway.assertion;

import java.util.Properties;

public class ExtendedAuthenticationAssertion extends
AuthenticationAssertionImpl {

   protected Properties m_properties = null;

   /**
    * @param id AuthenticationAssertion ID
    */
   public ExtendedAuthenticationAssertion(String id) {
      super(id);
   }

   /**
    * @return the m_properties
    */
   public Properties getProperties() {
      return m_properties;
   }

   /**
    * @param p_properties the properties to set
    */
   protected void setProperties(Properties p_properties) {
      m_properties = p_properties;
   }

   /**
    * @param p_propname Name der Eigenschaft
    * @param p_propvalue Wert der Eigenschaft
    */
   public void setProperty(String p_propname, String p_propvalue) {
      Properties properties = new Properties(getProperties());
      properties.setProperty(p_propname, p_propvalue);
      setProperties(properties);
   }

   /**
    * @param p_propname Name der Eigenschaft
    * @return Wert der Eigenschaft
    */
   public String getProperty(String p_propname) {
      return getProperties().getProperty(p_propname);
   }
}
```

F SessionInterceptor (Client)

```
/**
 * Der SessionInterceptor nutzt den SSOService,
 * um clientseitig ein Ticket fuer die aktuelle Sitzung zu erhalten
 *
 * @author mrichter; angepasst durch pemmann
 * @version 2009.10.05
 */

package de.delta.lm.client.cor.interceptor;

import java.util.ArrayList;
import java.util.Collection;

import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;

import de.delta.lm.client.ISessionIdHolder;
import de.delta.lm.cor.to.CommandTO;
import de.delta.lm.cor.to.ResponseTO;
import de.delta.lm.server.session.to.LoginResultTO;
import de.delta.sso.ISSOService;

public class SessionInterceptor implements MethodInterceptor {

  private static final String SID_KEY = "SID";
  private static final Collection<String> CREATE_SESSION_METHODS;
  private static final Collection<String> LOGOUT_METHODS;
  static{
    CREATE_SESSION_METHODS = new ArrayList<String>();
    CREATE_SESSION_METHODS.add("ISessionService.login");
    CREATE_SESSION_METHODS.add("ISessionService.createSession");

    LOGOUT_METHODS = new ArrayList<String>();
    LOGOUT_METHODS.add("ISessionService.logout");
  }

  private String m_sid = null;
  private ISessionIdHolder m_sessionIdHolder;

  @Autowired
  @Qualifier("session.ISSOService")
  private ISSOService m_ssoService;

  public Object invoke(MethodInvocation p_arg0) throws Throwable {
    CommandTO cmd = (CommandTO)p_arg0.getArguments()[0];

    if ( cmd.getHeaderParam(ISSOService.SSO_KEY) == null ) {
      cmd.addHeaderParam(ISSOService.SSO_KEY, m_ssoService.getCredentialId());
    }

    if(!isCreateSession(cmd) ){
      if(m_sessionIdHolder != null){
        cmd.addHeaderParam(SID_KEY, m_sessionIdHolder.getSessionId());
      }else{
        cmd.addHeaderParam(SID_KEY, m_sid);
      }
    }

    ResponseTO res = (ResponseTO)p_arg0.proceed();
```

```
    if(ResponseTO.EXCEPTION.equals(res.getResponseType())){
      //event. gucken ob sid abgelaufen
    }else{
      if(isCreateSession(cmd)){
        if(m_sessionIdHolder != null){
          m_sessionIdHolder.setSessionId(getSessionId(res));
        }else{
          m_sid = getSessionId(res);
        }
      } else if (isLogoutMethod(cmd)) {
        if(m_sessionIdHolder != null){
          m_sessionIdHolder.setSessionId(null);
        }else{
          m_sid = null;
        }
      }
    }
    return res;
  }

  private boolean isCreateSession(CommandTO p_cmd){
    return CREATE_SESSION_METHODS.contains(
       p_cmd.getObjectName()+"."+p_cmd.getMethodName());
  }

  private boolean isLogoutMethod( CommandTO p_cmd ) {
    return LOGOUT_METHODS.contains(
       p_cmd.getObjectName()+"."+p_cmd.getMethodName());
  }

  private String getSessionId(ResponseTO p_response){
    Object res = p_response.getResponseData();
    if(res instanceof LoginResultTO){
      return ((LoginResultTO)res).getSessionId();
    }else if(res instanceof String){
      return (String)res;
    }
    return null;
  }

  public void setSessionIdHolder(ISessionIdHolder p_sessionIdHolder) {
    m_sessionIdHolder = p_sessionIdHolder;
  }

}
```

G SSOInterceptor (Server)

```
/**
 * Der SSOInterceptor kontrolliert, ob ein Ticket übergeben wird. Aus diesem
 * stellt er die Leasman-Session-ID (LMS_SID) wieder her
 *
 * @author pemmann
 * @version 2009.10.05
 */

package de.delta.lm.server.cor.interceptor;

import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;

import de.delta.lm.cor.to.CommandTO;
import de.delta.lm.server.session.ISessionService;
import de.delta.sso.ISSOService;

public class SSOInterceptor implements MethodInterceptor
{
  private static final String LMS_SID = "SID";
  @Autowired @Qualifier("ISessionService")
  private ISessionService m_sessionService;
  @Autowired @Qualifier("session.ISSOService")
  private ISSOService m_ssoService;

  public Object invoke(MethodInvocation p_method) throws Throwable
  {
    CommandTO cto = null;
    Object obj = null;
    String sid = null;

    Object[] args = p_method.getArguments();
    cto = (CommandTO) args[0];
    String credentialId = (String) cto.getHeaderParam(ISSOService.SSO_KEY);

    try {
      sid = (String) m_ssoService.getCredentialProperty(
        credentialId, LMS_SID);

      if (sid == null) {
        // neue SessionID mittels "mdnr" und "bdnr" des Credential-Users
        sid = m_sessionService.createSession(
            Long.parseLong(m_ssoService.getUserAttr( credentialId, "mdnr" )),
            Long.parseLong(m_ssoService.getUserAttr( credentialId, "bdnr" )));

        // SessionID an Credential haengen
        m_ssoService.setCredentialProperty( credentialId, LMS_SID, sid );
      }

      cto.addHeaderParam(LMS_SID, sid);
      obj = p_method.proceed();

    } catch (Exception e) {       // kein gueltiges Token vorhanden
      e.printStackTrace();
    }

    return obj;
  }
}
```

H SSOTestCase

```
/**
 * Der SSOTestCase wird verwendet, um die Integration des SSOService zu testen
 *
 * @author pemmann
 * @version 2009.10.05
 */

package de.delta.lm.test.local.partner;

import java.util.Collection;
import java.util.Iterator;

import org.junit.Test;

import de.delta.common.exception.SystemException;
import de.delta.lm.server.partner.IPartnerService;
import de.delta.lm.server.partner.to.AnredeTO;
import de.delta.lm.test.local.LocalBlsBasicTestCase;

public class SSOTestCase extends LocalBlsBasicTestCase
{
  @Test
  public void ssoTestCase() throws SystemException
  {
    IPartnerService partnerService =
   (IPartnerService)m_factory.getBean("IPartnerService");

    Collection<AnredeTO> anreden = partnerService.getAllSalutations();

    Iterator<AnredeTO> it = anreden.iterator();

    while(it.hasNext()) {
      AnredeTO ato = it.next();
      System.out.print("[" + ato.getAnredeId() + ", " + ato.getBeschreibung()
   + "] ");
    }
  }

}
```

Literaturverzeichnis

[1pw] **Die richtige Single sign-on-Strategie, Nur noch ein Passwort**.
URL: http://www.searchsecurity.de/themenbereiche/applikationssicherheit/web-application-security/articles/62644/
[Stand 30.12.2009]

[ado] **Kerberos**.
URL: http://www.adopenstatic.com/images/resources/blog/Kerberos1.jpg
[Stand 20.11.2009]

[api] **OpenSSO Public API Documentation**.
URL: http://docs.sun.com/source/820-3739/index.html
[Stand 30.12.2009]

[bdo] **BLS API Dokumentation**. DELTA proveris AG, 2009.

[bds] **Bundesdatenschutzgesetz (Auszug) §9**.
URL: http://www.presserat.info/136.0.html
[Stand 30.12.2009]

[bls] **BLS Demo Client**. DELTA proveris AG, 2008.

[cmp] **Learn how to implement the Command pattern in Java**.
URL: http://www.javaworld.com/javaworld/javatips/jw-javatip68.html?page=1
[Stand 30.12.2009]

[dpb] **BLS - Die neue Mittelschicht**. DELTA proveris AG.
URL: http://www.depag.de/448.html
[Stand 30.12.2009]

[it] **IT-Wissen Lexikon**.
URL: http://www.itwissen.info/definition/lexikon/Identity-Federation-identity-federation.html
[Stand 30.12.2009]

[jaa] **Der Java Authentication and Authorization Service (JAAS)**.
URL: http://www.dpunkt.de/java/Programmieren_mit_Java/Sicherheit/14.html
[Stand 20.05.09]

[jbo] **Identity Management Made Easy with OpenSSO**.
URL: http://javaboutique.internet.com/tutorials/openSSO/
[Stand 28.12.2009]

[jkr] **JAAS Tutorial, Single Sign-On mit Kerberos**.
URL: http://www.oio.de/public/java/jaas/sso-jaas-kerberos-tutorial.htm
[Stand 20.05.2009]

[jma] Bruce Sams, Jakob Külzer: **Java Magazin**. 3/2008- Frankfurt a. M.:
Software & Support Verlag, 2008

[jnd] **Java Naming and Directory Interface (JNDI)**. URL: http://www.dpunkt.de/java/Programmieren_mit_Java/Java_Naming_and_Directory_Interface_(JNDI)/1.html
[Stand 20.05.2009]

[jsa] **JOSSO Architecture Overview**. URL:
http://www.josso.org/confluence/download/attachments/2949140/Architecture+Overview.jpg
[Stand 28.12.2009]

[krb] **Das Kerberos-Protokoll**.
URL: http://archiv.tu-chemnitz.de/pub/2006/0034/data/html/node12.html
[Stand 30.12.2009]

[kry01b] Burnett, Steve; Paine, Stephen: **Kryptographie – RSA Security's Official Guide**.
Deutsche Ausgabe - Bonn: mitp-Verlag, 2001

[kus] **Kerberos als Unternehmsweites Single-Sign-On**.
URL: http://www.mwiesner.com/wp-content/uploads/2007/06/lt2007kerberospaper.pdf
[Stand 01.01.2010]

[ldp03] Klünter, Dieter; Laser, Jochen: **LDAP verstehen, OpenLDAP einsetzen: Grundlagen, Praxiseinsatz, Single-Sign-on-Mechanismen**. - Heidelberg: dpunkt-Verl., 2003

[lma] **DELTA proveris AG**.
URL: http://typo.depag.de/DELTA-proveris-AG.906.0.html
[Stand 30.12.2009]

[lmk] **Leasman L15 - Konzeption**. DELTA proveris AG, 2006.

[mig] **Migrationsleitfaden Version 3.0 – Leitfaden für die Migration von Software**. URL:
http://www.cio.bund.de/cae/servlet/contentblob/294268/publicationFile/4678/migrationsleitfaden_download.pdf
[Stand 30.12.2009]

[osd] **OpenSSO**.
URL: https://opensso.dev.java.net/de
[Stand 29.12.2009]

[oso] **Fast and Free SSO: A Survey of Open-Source Solutions to Single Sign-On**. URL:
http://developers.sun.com/learning/javaoneonline/2007/pdf/TS-4604.pdf
[Stand 30.12.2009]

[oss] **OpenSSO: Session Service Architecture**.
URL: https://opensso.dev.java.net/files/documents/3676/23331/session_arch.pdf
[Stand 30.12.2009]

[pea] **Single Sign-On**.
URL: http://www.peak-solution.de/upload_material/223.pdf
[Stand 20.05.09]

[prg03] Hunt, Andrew; Thomas, David: **Der Pragmatische Programmierer**. - München:
Carl Hanser Verlag, 2003

[sam] **Sun Java System Access Manager FAQ**.
URL: http://developers.sun.com/identity/overview/faq/sso.jsp#3
[Stand 28.12.2009]

[spr] Eberhard Wolff: **Spring - Framework für die Java-Entwicklung**.
1. Aufl.- Heidelberg: dpunkt-Verl., 2006

[srv02] Holzmann, Jörg; Plate, Jürgen: **Linux-Server für Intranet und Internet**.
2., aktualisierte und erw. Aufl.- München: Carl Hanser Verlag, 2002

[sso] **SSO frei Haus – Einfache Lösungen zur Implementierung von Single Sign-On**. URL:
http://entwickler.de/zonen/portale/psecom,id,101,online,910,p,0.html
[Stand 30.12.2009]

[ssv] **Vergleich von Java SSO Lösungen**.
URL: http://www.oio.de/public/java/sso/single-sign-on-vergleich.htm
[Stand 28.12.2009]

[tec] **OpenSSO Technical Overview**.
URL: http://dlc.sun.com/pdf/820-3740/820-3740.pdf
[Stand 30.12.2009]

[win01] Eisenkolb, Kerstin; Gökhan, Mehmet: **Windows-Sicherheit**. - München:
Addison-Wesley Verlag, 2001

Selbständigkeitserklärung

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Bearbeitungsort, Datum	Unterschrift