Einführung eines Managementsystems zur Verwaltung von Benutzern, Rollen und Rechten in JDOSecure

Matthias Merz und Michael Grauer

Arbeitspapier 1/2006
April 2006

Arbeitspapiere in der Wirtschaftsinformatik

Matthias Merz und Michael Grauer
Universität Mannheim
Lehrstuhl für Wirtschaftsinformatik III
Schloss, D-68131 Mannheim
merz@uni-mannheim.de

Einführung eines Managementsystems zur Verwaltung von Benutzern, Rollen und Rechten in JDOSecure

Matthias Merz und Michael Grauer
Universität Mannheim
Lehrstuhl für Wirtschaftsinformatik III
Schloss, D-68131 Mannheim
merz@uni-mannheim.de

13. April 2006

Zusammenfassung

JDOSecure ist eine Sicherheitsarchitektur die den Zugriff auf persistente Objekte über die JDO-API kapselt und JDO um ein rollenbasiertes Berechtigungskonzept erweitert. Die zur Authentifizierung und Autorisierung notwendigen Daten werden über ein Benutzer-, Rollen- und Rechte-Managementsystem verwaltet. Dieses gestattet die Authentifizierung von Benutzern über ein Pluggable Authentication Module sowie die Verwaltung der Authentifizierungs- und Autorisierungsdaten in einer eigenständigen JDO-Ressource. Eine Anwendung zur Administration mit grafischer Benutzeroberfläche soll zudem die Verwaltung dieser Daten vereinfachen.

1 Einführung

Java Data Objects (JDO) ist ein Industriestandard für Objekt-Persistenz und ermöglicht Java-Entwicklern einen transparenten Umgang mit persistenten Objekten (Java Community Process 2004a). Die JDO-Spezifikation definiert hierzu eine datenbankunabhängige Schnittstelle (JDO-API) und stellt neben der Austauschbarkeit des verwendeten Datenbanksystems auch die Substituierbarkeit unterschiedlicher JDO-Implementierungen untereinander sicher.

JDOSecure[1] erweitert JDO um ein rollenbasiertes Berechtigungskonzept (vgl. Merz 2005). Hierzu kapselt JDOSecure den Zugriff auf persistente Objekte über die JDO-API. Berechtigungen lassen sich mit JDOSecure in Abhängigkeit eines Pakets bzw. einer Klasse definieren und individuell an eine Benutzer-Rolle sowie an unterschiedliche *CRUD*[2]-Operation binden.

Zur Authentifizierung und Autorisierung des Anwenders nutzt JDOSecure den *Java Authentication and Authorization Service* (*JAAS*). Hierbei erfolgt die Zuweisung der Rechte über die Definition von *Permissions* in entsprechenden *Policy*-Dateien. Da mit steigender Zahl an Benutzer-Rollen und Rechten die Administration aufwändig, zunehmends unübersichtlich und infolgedessen auch fehleranfällig wird, wurde JDOSecure um ein Benutzer-, Rollen- und Rechte-Managementsystem erweitert. Dieses regelt die Auslagerung der Daten zur Authentifizierung und Autorisierung in einer separaten JDO-Ressource und vereinfacht das Management von Benutzern, Rollen und Rechten über eine Administrationsanwendung mit grafischer Benutzeroberfläche.

[1] Weitere Informationen zu JDOSecure sind unter `http://projekt-jdo.uni-mannheim.de/JDOSecure` abrufbar.

[2] CRUD steht als Akronym für die grundlegenden Operationen Create, Retrieve, Update und Delete. Bei der Realisierung von Objektpersistenz werden durch ein Persistenz-Framework i.d.R. entsprechende Methoden zur Erzeugung, Anfrage, Veränderung und dem Löschen von persistenten Objekten zur Verfügung gestellt.

In diesem Artikel wird das Benutzer-, Rollen- und Rechte-Managementsystem als Erweiterung von JDOSecure vorgestellt. Hierzu werden im nächsten Abschnitt zunächst die sicherheitstechnisch relevanten Grundlagen wiederholt. Anschließend werden die Konzepte der Sicherheitsarchitektur JDOSecure skizziert. In Abschnitt 4 wird ferner die Architektur des Benutzer-, Rollen- und Rechte-Managementsystems vorgestellt sowie dessen Anbindung an JDOSecure verdeutlicht. Nachfolgend wird das Datenmodell erläutert und die Anwendung zur Administration von Benutzern, Rollen und Rechten vorgestellt. Im letzen Abschnitt werden die noch offenen Punkte erörtert.

2 Authentifizierung und -autorisierung in Java

Der *Java Authentication and Authorization Service* (*JAAS*) gestattet die benutzerorientierte Authentifizierung und Autorisierung beim Zugriff auf Systemressourcen (vgl. Sun Microsystems 2001, Sun Microsystems 2005, Gong 2002). Der `AccessController` prüft hierzu die Rechte eines authentifizierten Benutzers vor Ausführung sicherheitsrelevanter Operationen, wie z. B. dem Zugriff auf das Dateisystem, dem Setzen und Lesen von Systemproperties oder der Initiierung einer Netzwerkkommunikation über Sockets. JDOSecure und das in Abschnitt 4 vorgestellte Benutzer-, Rollen- und Rechte-Managementsystem bauen auf JAAS auf, weshalb auf die Authentifizierung und Autorisierung mittels JAAS in diesem Abschnitt näher eingegangen wird.

2.1 Authentifizierung

Die Authentifizierung bezeichnet den Vorgang, die Identität eines Anwenders mit Hilfe bestimmter Merkmale zu überprüfen. JAAS ist hierbei nicht auf einfache Passwortverfahren beschränkt. Über die Einbindung so genannter *Pluggable Authentication Modules* (*PAM*) lässt sich die Authentifizierung auch an externe Dienste delegieren (Sun Microsystems 2001, Samar, Vipin und Lai, Charlie 1996, Open Software Foundation 1995). Über eine Konfigurationsdatei lassen sich entsprechende *Login*-Module definieren (Sun Microsystems 2002) und über diese festlegen, ob die Authentifizierung zur Laufzeit beispielsweise mit Kerberos oder LDAP erfolgen soll.

Wie im JAAS Reference Guide beschrieben, erzeugt eine JAAS-nutzende Anwendung zunächst eine `LoginContext`-Instanz (Sun Microsystems 2001). Diese ermittelt über das Auswerten einer Konfigurationsdatei welche Login-Module zur Authentifizierung genutzt werden sollen. Die Module verwenden `CallbackHandler` zur Kommunikation mit der Anwendung, der Umgebung oder dem Anwender, beispielsweise zur Abfrage einer Benutzerkennung und eines Passworts. Während des Login-Vorgangs wird ein Objekt der Klasse `Subject` konstruiert, welches den für diese Sitzung authentifizierten Benutzer repräsentiert. Ist der Login-Vorgang erfolgreich, so werden dem `Subject` verschiedene `Principal`-Objekte und *Credentials* zugewiesen. Ein `Principal` stellt hierbei eine *Identität*[3] des Benutzers dar (Oaks 2001, Java Community Process 2004b). *Credentials* bezeichnen sicherheitsbezogene Attribute wie öffentliche oder private Schlüssel. Hierfür gibt es keine konkrete Java-Klasse; jedes Objekt kann ein Credential darstellen (Sun Microsystems 2001).

2.2 Autorisierung

Ist die Identität eines Anwenders erfolgreich festgestellt, folgt die Überprüfung von Berechtigungen in der Phase der Autorisierung. Vor einem Zugriff auf eine Systemressource werden die Rechte eines Benutzers vom `SecurityManager` geprüft, der diese Aufgabe an den `AccessController`

[3]Der Begriff der Identität ist weiter gefasst als der Begriff einer Rolle, schließt diese jedoch mit ein. Eine Identität kann verstanden werden als *eine* identifizierende Charakteristik des Benutzers.

delegiert. Zur Laufzeit bezieht dieser sämtliche, einem Benutzer (`Subject`) zugewiesene Identitäten in den Prozess der Berechtigungsprüfung mit ein und gewährt oder verweigert entsprechend den Zugriff.

Die einem Benutzer gewährten Rechte werden über die Instanz der Klasse `Policy` realisiert. Sie ermittelt und liefert die Rechte des momentan authentifizierten Benutzers auf Basis seiner `Principal`-Objekte (Benutzer-Rollen) (Sun Microsystems 2001). Die Definition dieser Rechte erfolgt hierbei über *Policy*-Dateien. In der Sicherheitsarchitektur von Java kann in einer *Java Virtual Machine* (*JVM*) zu jedem Zeitpunkt maximal eine Instanz einer `Policy`-Klasse aktiv sein. Die aktive Instanz kann über eine Anweisung im Programmcode oder durch einen Eintrag in der Datei *java.security* gesetzt werden (Gong 2002, Abschnitt 3.3). Neben `Principal`-Objekten existieren Objekte der Klasse `CodeSource`. Diese spezifizieren die Adresse, von der auszuführender Programmcode geladen wurde (vgl. Sun Microsystems 2001). Dies ermöglicht es, Rechte an Programmcode zu binden und so verschiedene Anwendungen mit unterschiedlichen Rechten auszustatten.

3 JDOSecure: Eine Sicherheitsarchitektur für die JDO-Spezifikation

In diesem Kapitel werden die Konzepte der Sicherheitsarchitektur JDOSecure zur Authentifizierung und Autorisierung erläutert und die Möglichkeiten der Integration mit einer JDO-Implementierung skizziert.

3.1 Das Konzept von JDOSecure

Die Konstruktion einer `PersistenceManagerFactory`-Instanz erfolgt in der Regel über die statische Methode `getPersistenceManagerFactory(Properties props)` der `JDOHelper`-Klasse. Die Verwendung dieser Methode ermöglicht den Austausch einer JDO-Implementierung, ohne dass hierzu die Modifikation des Java-Quellcodes einer Anwendung notwendig wird.

Um die hier vorgestellte Sicherheitsarchitektur unabhängig von der konkret verwendeten JDO-Implementierung einsetzen zu können, baut JDOSecure auf diesem Konzept auf. Eine von `JDOHelper` abgeleitete `JDOSecureHelper`-Klasse bildet den Einstiegspunkt für Anwendungen zur Nutzung der Sicherheitsarchitektur. Hierzu überschreibt diese Klasse die `getPersistenceManagerFactory()`-Methode (vgl. Abbildung 1). Anhand der übergebenen Benutzerkennung und des Passworts nimmt JDOSecure die Authentifizierung des Anwenders vor. Die hierzu notwendigen Daten werden über das in Abschnitt 4 beschriebene Benutzer-, Rollen- und Rechte-Managementsystem in einer separaten JDO-Ressource verwaltet.

Nach erfolgreicher Authentifizierung wird eine `PersistenceManagerFactory`-Instanz erzeugt und der Anwendung zur Verfügung gestellt[4]. Bevor die JDO-Implementierung die Konstruktion dieser Instanz vornimmt, substituiert JDOSecure die Benutzerkennung und das Passwort durch Vorgaben, die dem Benutzer bzw. der Anwendung nicht bekannt sind. Dies soll den Aufbau einer direkten Verbindung zum Persistenzmedium unter Umgehung von JDOSecure verhindern.

Zur Realisierung der Authentifizierung wird der Java Authentication and Authorization Service (Sun Microsystems 2001) verwendet. Abbildung 2 veranschaulicht die Umsetzung der Authentifizierung mit Hilfe eines vereinfachten UML-Klassendiagramms. Über den Methodenaufruf `getPersistenceManagerFactory()` der `JDOSecureHelper`-Klasse wird zunächst eine `LoginContext`-Instanz erzeugt. Diese leitet die Authentifizierung über ein `JDOLoginModule` an einen `JDOCallbackHandler` weiter. Dieser nutzt die in dem `Properties`-Objekt übergebene Benutzerkennung sowie das Passwort zur Authentifizierung. Sofern dieser Vorgang ohne

[4]Wie in Abschnitt 3.2 beschrieben, wird der Anwendung anstelle einer `PersistenceManagerFactory`-Instanz ein Proxy-Objekt zurückgeliefert. Dies hat an dieser Stelle jedoch keine Auswirkung.

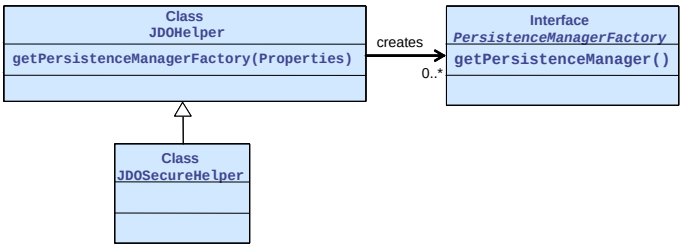

Abbildung 1: Klassendiagramm zur Veranschaulichung der Integration der JDOSecureHelper Klasse in die JDO-API

`SecurityException` abgeschlossen wird, bezieht die `JDOSecureHelper`-Instanz über den `LoginContext` den authentifizierten `JDOUser`.

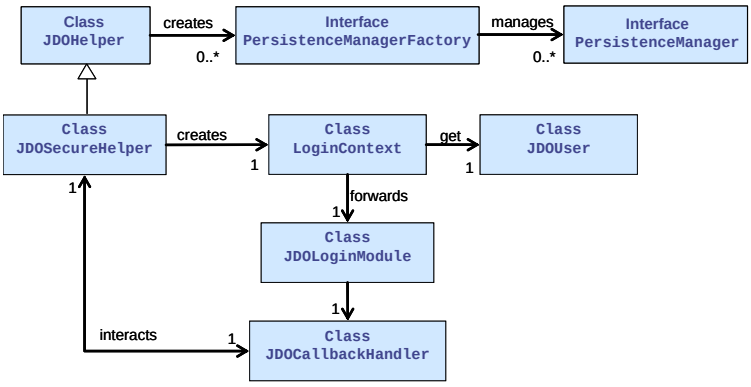

Abbildung 2: Realisierung der Authentifizierung

Nach erfolgreicher Authentifizierung folgt die Phase der Autorisierung. Zur Laufzeit einer Anwendung wird in dieser geprüft, ob ein authentifizierter Benutzer über die notwendigen Berechtigungen zur Ausführung von CRUD-Operationen auf `PersistenceCapable`-Instanzen verfügt. Hierzu lassen sich die Rechte individuell an ein bestimmtes Paket bzw. eine Klasse binden oder beispielsweise für die Rolle eines Administrators mit Hilfe des *Wildcard*-Ausdrucks ¨`*`¨ auf alle persistenten Klassen ausweiten.

Die Überprüfung der Berechtigungen eines Benutzers wird hierbei über die Methoden der Klasse `PersistenceManager` wie z. B. `makePersistent()`, `newQuery()` und `deletePersistent()` realisiert. Zur Ausführung einer solchen Methode sind entsprechende Permissions notwendig, die JDOSecure in Abhängigkeit der konkreten `PersistenceCapable`-Instanz prüft. Die Rechte zur Persistenzierung von Instanzen, die dem Paket sample zugeordnet sind, lassen sich beispielsweise für einen Principal „sampleuser“ wie folgt definieren:

```
grant Principal JDOUser ¨sampleuser¨
        permission JDOMakePersistentPermission ¨sample.*¨;
}
```

Das Update einer `PersistenceCapable`-Instanzen ist nicht an eine spezielle Methode des `PersistenceManager`s gebunden. JDO folgt dem Konzept der transparenten Persistenz und die Aktualisierung von Objektattributen wird selbständig durch die JDO-Implementierung an das Persistenzmedium propagiert. Über eine Substitution des JDO-`StateManager`s mit einem *Dynamic Proxy* ist JDOSecure jedoch auch der Eingriff in Update-Vorgänge möglich.

Hierdurch wird eine Berechtigungsprüfung vor Aktualisierung von Objektattributen möglich und die Existenz einer entsprechenden `JDOUpdatePermission` geprüft.

Auf den Dynamic Proxy-Ansatz und die Kollaboration von JDOSecure und einer JDO-Implementation wird im nächsten Abschnitt näher eingegangen.

3.2 Integration der JDOSecure-Sicherheitsarchitektur

Wesentliche Voraussetzung für die Akzeptanz der vorgestellten Sicherheitsarchitektur ist die Unabhängigkeit von einer konkreten JDO-Implementation. Wichtige Nebenbedingung hierbei ist, dass ein allgemeiner Lösungsansatz keine Modifikation der JDO-API erfordert oder eine Anpassung des Enhancement-Vorgangs impliziert.

Eine Möglichkeit, mit der sich die vorgestellte Sicherheitsarchitektur in Java entsprechend umsetzen lässt, ist das Dynamic Proxies-Konzept. Dynamic Proxies gestatten die Erzeugung einer Proxy-Instanz[5] dynamisch zur Laufzeit (Blosser 2000). Zur Erzeugung eines solchen Proxies wird die statische Methode `newProxyInstance()` der Klasse `java.lang.reflect.Proxy` aufgerufen, welcher ein `ClassLoader`-Objekt, ein Array mit Interfaces sowie ein `InvocationHandler`-Objekt übergeben werden. Eingehende Anfragen werden von der erzeugten Proxy-Instanz an den `InvocationHandler` weitergeleitet, der diese je nach Implementierung entweder selbst bearbeitet oder an das ursprüngliche Objekt delegiert.

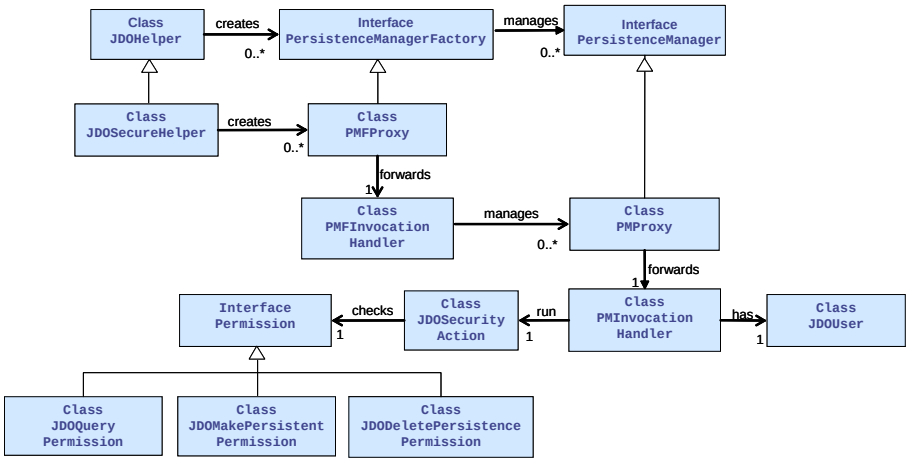

Abbildung 3: Realisierung der Autorisierung

Die Einbindung der vorgestellten Sicherheitsarchitektur in JDO wird, wie Abbildung 3 verdeutlicht, über das Konzept der Dynamic Proxies realisiert. Ausgehend von der `JDOSecureHelper`-Klasse bezieht ein Anwender nach Aufruf der `getPersistenceManagerFactory()`-Methode anstelle einer `PersistenceManagerFactory`-Instanz ein korrespondierendes Proxy-Objekt. Über den `PMFInvocationHandler` besteht in diesem Fall die Möglichkeit, Methodenaufrufe, die ursprünglich an die `PersistenceManagerFactory`-Instanz gerichtet sind, abzufangen und gezielt zu manipulieren. JDOSecure nutzt diese Möglichkeit und stellt sicher, dass bei Aufruf der Methode `getPersistenceManager()` dem Anwender anstelle eines `PersistenceManager`-Objekts eine weitere Proxy-Instanz zurückgeliefert wird. Diese Proxy-Instanz leitet auch hier sämtliche Methodenaufrufe, die in diesem Fall an den `PersistenceManager` gerichtet sind, an einen

[5]Ein Proxy bezeichnet ein Stellvertreter-Objekt, das einem anderen Objekt vorgelagert ist. Der Zugriff auf das original Objekt erfolgt ausschließlich über den Proxy, der eingehende Anfragen an dieses weiterleitet. In Java lassen sich solche *statischen Proxies* durch Klassen realisieren, welche die Schnittstellen des original-Objekts implementieren.

`PMInvocationHandler` weiter, über den JDOSecure die Prüfung der rollenbasierten Rechten realisiert.

Der `PMInvocationHandler` kann infolgedessen für einen zuvor authentifizierten `JDOUser` prüfen, ob dieser über die Rechte zur Ausführung der entsprechenden Methode und einer übergebenen Klasse oder eines konkreten Objekts verfügt. Intern wird dazu über eine `JDOSecurityAction`-Instanz die statische Methode `Subject.doAs(subject, action)` aufgerufen und die eigentliche Prüfung an den `AccessController` delegiert. Sofern der Anwender über die entsprechende Berechtigung verfügt, wird der Methodenaufruf an die ursprüngliche `PersistenceManager`-Instanz weitergeleitet. Andernfalls wird eine `JavaSecurity`-Exception ausgeworfen und der Vorgang abgebrochen.

Die Zuweisung von Berechtigungen erfolgt in der Regel über die Konfiguration entsprechender Policy-Dateien. Da mit steigender Zahl an Benutzer-Rollen und Rechten die Administration aufwändig, zunehmends unübersichtlich und infolgedessen auch fehleranfällig wird, wurde JDOSecure um ein Benutzer-, Rollen- und Rechte- Managementsystem erweitert, das im nachfolgenden Abschnitt vorgestellt wird.

4 Benutzer-, Rollen- und Rechte-Managementsystem

In diesem Abschnitt wird das Benutzer-, Rollen- und Rechte-Managementsystem als Erweiterung von JDOSecure vorgestellt. Zunächst wird die Anbindung des Benutzer-, Rollen- und Rechte-Managementsystems an JDOSecure verdeutlicht und auf seine Architektur eingegangen. Nachfolgend wird das Datenmodell erläutert und die Anwendung zur Administration von Benutzern, Rollen und Rechten vorgestellt.

4.1 Anbindung des Benutzer-, Rollen- und Rechte-Managementsystems

Das Benutzer-, Rollen- und Rechte-Managementsystem regelt die Auslagerung der Daten zur Authentifizierung und Autorisierung in einer separaten JDO-Ressource und vereinfacht das Management von Benutzern, Rollen und Rechten über eine Administrationsanwendung mit grafischer Benutzeroberfläche.

Die Anbindung des Benutzer-, Rollen- und Rechte-Managementsystems an JDOSecure wird in Abbildung 4 skizziert. Der linke Teil der Abbildung umfasst die Komponenten der Anwendungsdomäne bestehend aus Benutzeroberfläche, Anwendung, JDOSecure, einer JDO-Implementation und einer JDO-Ressource zur Verwaltung persistenter Instanzen. Der rechte Teil der Abbildung zeigt den schematischen Aufbau des Benutzer-, Rollen- und Rechte-Managementsystems bestehend aus Administrationsanwendung, Benutzer-, Rollen- und Rechte-Verwaltung, JDO-Implementation und einer separaten JDO-Ressource zur Ablage der Verwaltungsdaten. Aus der Abbildung wird zudem deutlich, dass die Anbindung des Benutzer-, Rollen- und Rechte-Managementsystems mit JDOSecure auf Ebene der Anwendungsschicht erfolgt und sich diese für den (End-)Anwender als transparent darstellt.

Das UML-Komponentendiagramm in Abbildung 5 stellt die Anbindung detaillierter dar. Da JDOSecure auf JAAS aufbaut, wurden die abstrakte Java Klasse `Policy` und das JAAS Interface `LoginModule` als Schnittstellen ausgewählt: Die Klasse `JDOLoginModule` implementiert das Interface `LoginModule` und die Klasse `JDOPolicy` erweitert die abstrakte Klasse `Policy`. Objekte dieser Klassen bilden die einzigen Berührungspunkte zwischen dem Benutzer-, Rollen- und Rechte-Managementsystem und JDOSecure.

Die Klasse `JDOLoginModule` implementiert den PAM-Standard (das Java-Interface `LoginModule`) und ermöglicht so den Zugriff auf die zur Authentifizierung relevanten Daten wie Benutzerkennung und Passwort. Damit zur Authentifizierung des Benutzers eine Instanz der Klasse `JDOLoginModule` verwendet werden kann, muss die Konfigurationsdatei

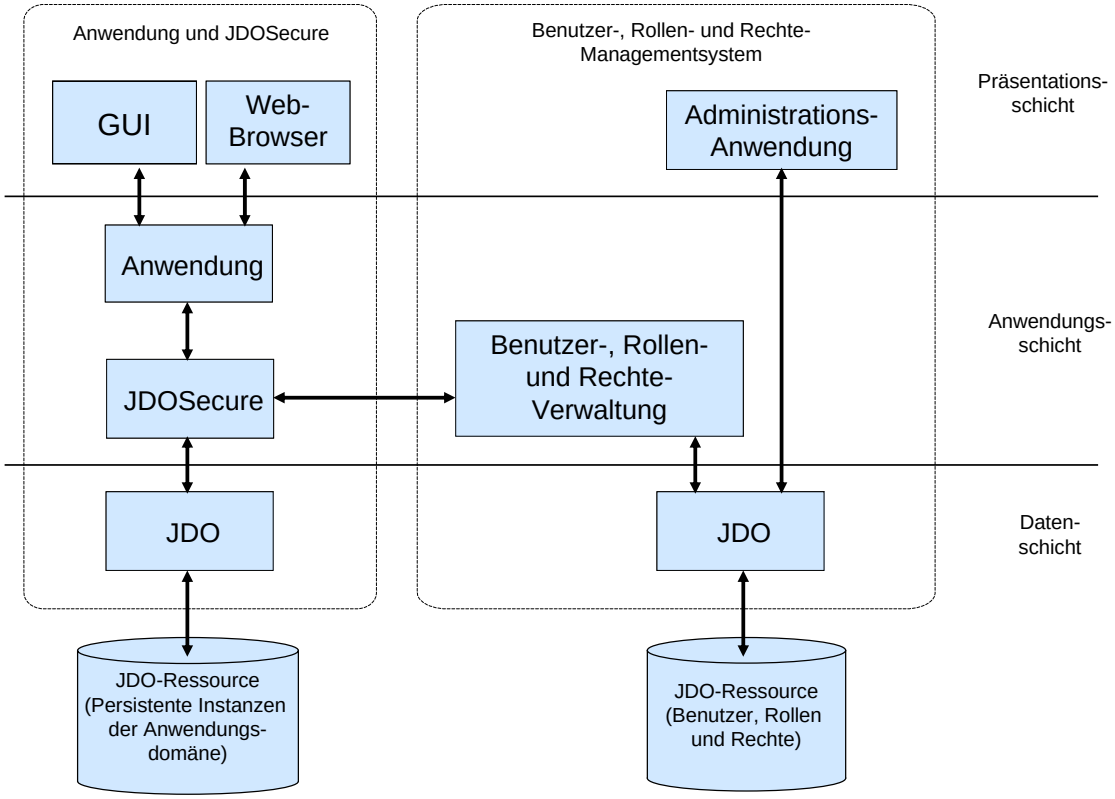

Abbildung 4: Anbindung des Benutzer-, Rollen- und Rechte-Managementsystems an JDOSecure

für Login-Module (*JAAS Login Configuration File*) folgenden Eintrag aufweisen (vgl. Sun Microsystems 2002):

```
JDOAccessControl {
   de.unimannheim.wifo.superman.plugin.authentication.JDOLoginModule required
   debug=false;
};
```

Der vom `JDOSecureHelper` erzeugte `LoginContext` ermittelt aus dieser Datei welches Login-Modul zur Authentifizierung verwendet werden soll. In Abhängigkeit der ausgeführten Anwendung sind verschiedene Einstellungen möglich; der Eintrag `JDOAccessControl` kennzeichnet die Einstellungen bei Ausführung von JDOSecure. Der Parameter `required` legt fest, dass eine Authentifizierung mittels einer Instanz der Klasse `JDOLoginModule` erfolgreich sein muss, um den Login-Vorgang des Benutzers abzuschließen. Der Eintrag `debug=false` unterdrückt Rückmeldungen, die andernfalls während des Vorgangs der Authentifizierung ausgegeben werden. Wie bereits in Abbildung 2 dargestellt übergibt der `JDOCallbackHandler` die Benutzerkennung und das Passwort an das `JDOLoginModule`. Befindet sich in der JDO-Ressource ein `Credential`-Objekt mit derselben Benutzerkennung und demselben Passwort, so ist die Authentifizierung erfolgreich und das `JDOLoginModule` übergibt die mit dem `Credential`-Objekt assoziierten `Principal`-Objekte an den `LoginContext`. Dieser fügt der aktuellen Sitzung (dem `Subject`) alle `Principal`-Objekte hinzu, womit die Benutzerauthentifizierung abgeschlossen ist.

Die Klasse `JDOPolicy` erweitert die standardmäßige `Policy`-Implementation von Java und gewährt Zugriff auf die Daten zur Autorisierung der Benutzer (`Principal`-Objekte, Rechte). In einer JVM ist zu jedem Zeitpunkt immer genau eine Instanz einer `Policy`-Klasse aktiv und kontrolliert die Zugriffsberechtigungen für die gesamte JVM. Damit die Klasse `JDOPolicy` die

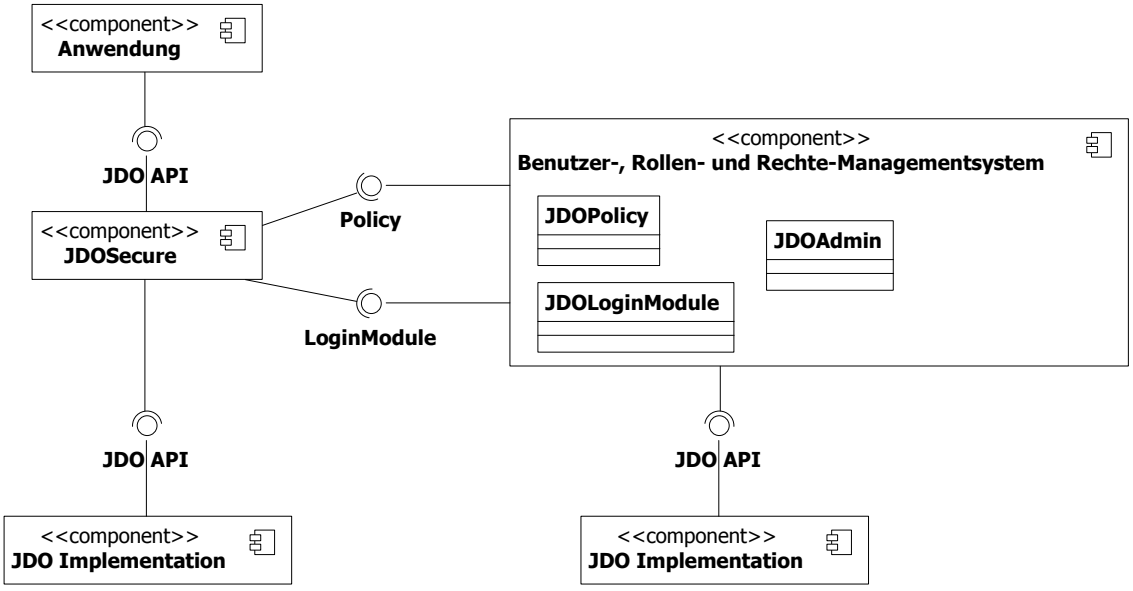

Abbildung 5: Komponenten des Benutzer-, Rollen- und Rechte-Managementsystem

Berechtigungsprüfung übernehmen kann, muss eine Instanz dieser Klasse zur Laufzeit erzeugt und aktiviert werden. Der nachfolgend dargestellte Programmcode kann z. B. innerhalb einer *Initialisierungsanwendung* verwendet werden, welche unmittelbar vor dem Start von JDOSecure ausgeführt wird:

```
JDOPolicy jp = new JDOPolicy(Policy.getPolicy());
Policy.setPolicy(jp);
```

Bei der Erzeugung einer Instanz der Klasse `JDOPolicy` wird die bislang aktive `Policy`-Instanz übergeben; `JDOPolicy` berücksichtigt diese bei allen Berechtigungsprüfungen: Wird eine Berechtigung von der bislang aktiven `Policy`-Instanz bestätigt, so wird sie auch von `JDOPolicy` bestätigt. Die bisher in *Policy*-Dateien definierten Berechtigungen sind also weiterhin gültig. Ab dem Zeitpunkt der Aktivierung wird `JDOPolicy` bei Berechtigungsprüfungen vom `SecurityManager` bzw. `AccessController` miteinbezogen. Hierbei werden die `Principal`-Objekte der aktuellen Sitzung übergeben, die `CodeSource`-Objekte des ausgeführten Programmcodes und ein Objekt der Java Klasse `Permission`, welches die zu prüfende Berechtigung darstellt. `JDOPolicy` ermittelt anhand der JDO-Ressource, mit welchen Rechten die übergebenen `Principal`- und `CodeSource`-Objekte verknüpft sind, und ob die zu prüfende Berechtigung durch diese Rechte abgedeckt ist.

Die abgebildete Klasse `JDOAdmin` repräsentiert die Administrationsanwendung, mit der Benutzer, Rollen und Rechte in der JDO-Ressource verwaltet werden. Hierbei unterstützt sie den Anwender durch ihre grafische Benutzeroberfläche und ermöglicht so eine komfortable Verwaltung. Auf die Administrationsanwendung und ihre Handhabung wir in Abschnitt 4.3 eingegangen.

Die Klassen `JDOLoginModule`, `JDOPolicy` und `JDOAdmin` sind weitestgehend voneinander unabhängig, sodass das Benutzer-, Rollen- und Rechte-Managementsystem z. B. auch nur zur Authentifizierung (`JDOLoginModule` und `JDOAdmin`) oder lediglich zur Autorisierung (`JDOPolicy` und `JDOAdmin`) eingesetzt werden kann. Das gemeinsame, verknüpfende Element ist das zugrunde liegende Datenmodell, das im folgenden Abschnitt vorgestellt wird.

4.2 Datenmodell

Das Datenmodell des Benutzer-, Rollen- und Rechte-Managementsystems wurde so einfach und flexibel wie möglich gehalten; konkrete Verwaltungsmethoden mit fest umrissenen Zuweisungsregeln werden erst durch die Administrationsanwendung verwirklicht. Das Ziel war, die Datenbasis von der eingesetzten Verwaltungsmethode unabhängig zu machen. So können Verwaltungsmethoden, die ihre Daten in Form des flexiblen Datenmodells speichern können, leicht ausgetauscht werden. Da das Datenmodell unverändert bleibt, kann die Benutzer-, Rollen- und Rechte-Verwaltung ebenfalls unverändert bleiben. Lediglich an der Administrationsanwendung müssen Änderungen bzw. Erweiterungen vorgenommen werden.

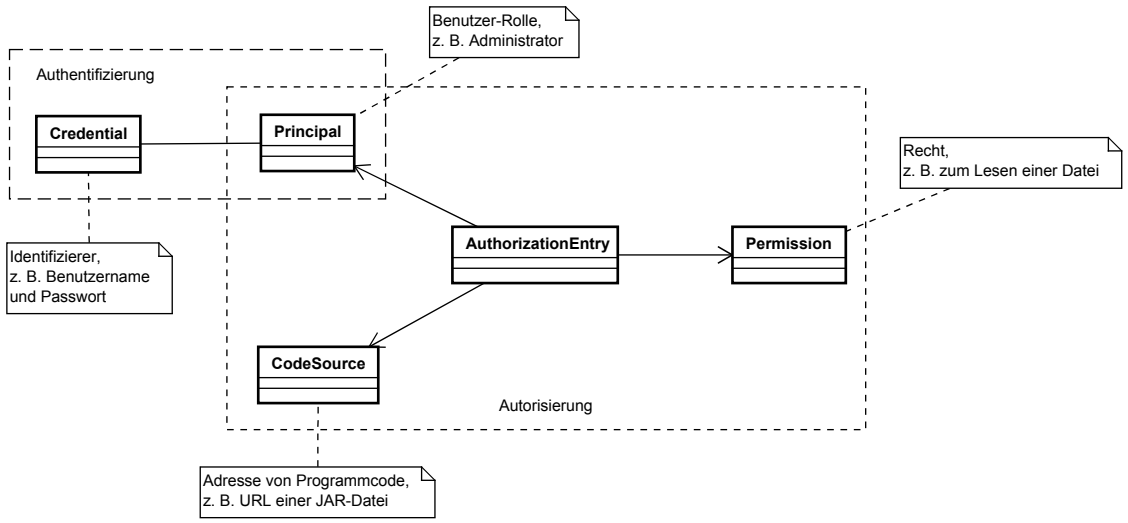

Abbildung 6: Datenmodell des Benutzer-, Rollen- und Rechte-Managementsystems

In Abbildung 6 ist das Datenmodell dargestellt; die beiden Aufgabengebiete *Authentifizierung von Benutzern* und *Autorisierung* heben sich deutlich ab. Das verbindende Element ist die Klasse `Principal`, welche der Klasse in JAAS entspricht und eine Identität eines Benutzers repräsentiert.

Die Klasse `Credential` stellt eine Identifikation eines Benutzers dar. Ein Benutzer kann auf mehrere Arten identifiziert werden; das Benutzer-, Rollen- und Rechte-Managementsystem verwendet Benutzerkennung und Passwort.

Der Benutzer selbst ist im Datenmodell nicht explizit repräsentiert: Es werden lediglich alle den Benutzer identifizierenden `Credential`-Objekte gespeichert sowie alle seine Identitäten (`Principal`-Objekte). Der Benutzer ist implizit, durch die Verknüpfungen zwischen diesen `Credential`- und `Principal`-Objekten gespeichert.

Rechte werden durch die Java Klasse `Permission` dargestellt. Objekte der Klasse `AuthorizationEntry` verknüpfen Rechte (`Permission`-Objekte) und Benutzer-Identitäten (`Principal`-Objekte), sodass die einer Identität zugewiesenen Rechte leicht ermittelt werden können. Welche Aktionen zugelassen sind, ergibt sich demnach aus der Menge der Rechte, die alle dem `Subject` zugewiesenen Identitäten zusammen besitzen.

Die Klasse `CodeSource` entspricht der gleichnamigen Java Klasse; mit ihr kann der Administrator Rechte an Programmcode binden. Da sich `Permission`-Objekte (durch Objekte der Klasse `AuthorizationEntry`) zugleich mit `Principal`- und `CodeSource`-Objekten verknüpfen lassen, ist eine differenzierte Zuweisung von Rechten möglich. So kann z. B. der Identität X Zugriff auf alle Dateien eines Verzeichnisses Y gewährt werden, wenn Anwendung A ausgeführt wird. Wird eine andere Anwendung ausgeführt, wird der Zugriff verweigert.

Im nächsten Abschnitt wird die Anwendung zur Administration von Benutzern, Rollen und Rechten mit grafischer Benutzeroberfläche vorgestellt.

4.3 Anwendung zur Administration von Benutzern, Rollen und Rechten

Wie bereits in Kapitel 4.2 deutlich wurde, ist das Datenmodell des Benutzer-, Rollen- und Rechte-Managementsystems aus Flexibilitätsgründen weitgehend allgemein gehalten. Daher ist es die Aufgabe der Administrationsanwendung, auf dem Datenmodell aufbauend eine komfortable Verwaltung zu realisieren. Die Administrationsanwendung basiert auf einer grafischen Benutzeroberfläche und stellt hierfür mehrere Verwaltungsmodi zur Verfügung:

Es existiert ein allgemeiner Verwaltungsmodus, bei dem unmittelbar auf dem Datenmodell operiert wird: `AuthorizationEntry`-Objekte werden direkt erzeugt und mit `Principal`-, `Permission`- und `CodeSource`-Objekten verknüpft. Die Möglichkeiten des flexiblen Datenmodells können in diesem Modus vollständig ausgeschöpft werden: So ist es möglich Rechte an Benutzer und Programmcode zugleich zu binden, wie im vorigen Abschnitt bereits erläutert; oder es können Rechte an eine *Menge* von `Principal`-Objekten gebunden werden, was bedeutet, dass das Recht nur vergeben wird, wenn alle `Principal`-Objekte dieser Menge vorliegen. Aufgrund dieser Mächtigkeit gestaltet sich die Verwaltung jedoch wenig übersichtlich und ist, werden Rechte an *Mengen* von `Principal`-Objekten gebunden, schwer zu durchschauen.

Ein zweiter Verwaltungsmodus orientiert sich daher am *Role-Based Access Control* (*RBAC*) Standard (vgl. Ferraiolo, David F., Sandhu, Ravi, Gavrila, Serban, Kuhn, D. Richard und Chandramouli, Ramaswamy 2001). Hierbei wird die Verwaltung von Code-Rechten (Rechte für `CodeSource`-Objekte) von der Verwaltung von Benutzer-Rechten (Rechte für `Principal`-Objekte) getrennt. Dem Benutzer (in diesem Modus repräsentiert durch die ihn identifizierenden `Credential`-Objekte) werden Benutzer-Rollen (`Principal`-Objekte) zugewiesen; jede von diesen Benutzer-Rollen wird wiederum mit Rechten (`Permission`-Objekte) verknüpft. Dieser Verwaltungsmodus ist gut zu überschauen – die Menge der Rechte, die ein Benutzer hat, ist einfach die Summe der Rechte seiner Benutzer-Rollen. Diese Übersichtlichkeit hat jedoch ihren Preis: Es ist nicht mehr möglich einem Benutzer unterschiedliche Rechte in Abhängigkeit des Programmcodes zu gewähren. Auch können Berechtigungen immer nur für eine Benutzer-Rolle definiert werden (nicht für Mengen von Benutzer-Rollen), was u. U. die Einführung zusätzlicher Rollen nötig macht.

Abbildung 7 zeigt die grafische Oberfläche der Administrationsanwendung: Der an RBAC orientierte Verwaltungsmodus ist gewählt, und es ist die Registerkarte im Vordergrund, mit der Rechte an Benutzer-Rollen geknüpft werden können. Das kleine Dialogfenster im Vordergrund dient zur Definition neuer Berechtigungen und erstellt ein neues `Permission`-Objekt. Die rechte Liste im großen Fenster zeigt alle bereits definierten Rechte. Die linke Liste enthält alle vorhandenen Benutzer-Rollen (`Principal`-Objekte). Ist in der linken Liste eine Benutzer-Rolle selektiert, so werden in der mittleren Liste die Rechte angezeigt, die dieser Benutzer-Rolle zugewiesen sind. Durch die Buttons mit den Pfeilen können Rechte zugewiesen oder entfernt werden. Die Buttons mit Plus oder Minus definieren und löschen neue Benutzer-Rollen oder Rechte. Auch die restliche Verwaltung ist nach diesem Prinzip der *Verknüpfung zweier Mengen* aufgebaut, sodass sich die Handhabung der Administrationsanwendung durchgängig gleich und einfach gestaltet.

Da die Administrationsanwendung von den restlichen Programmteilen weitgehend unabhängig ist, können die sicherheitsrelevanten Daten auch von einem anderen Rechner aus verwalten werden. Einzige Voraussetzung hierfür ist, dass die genutzte JDO-Ressource einen entfernten Zugriff gestattet.

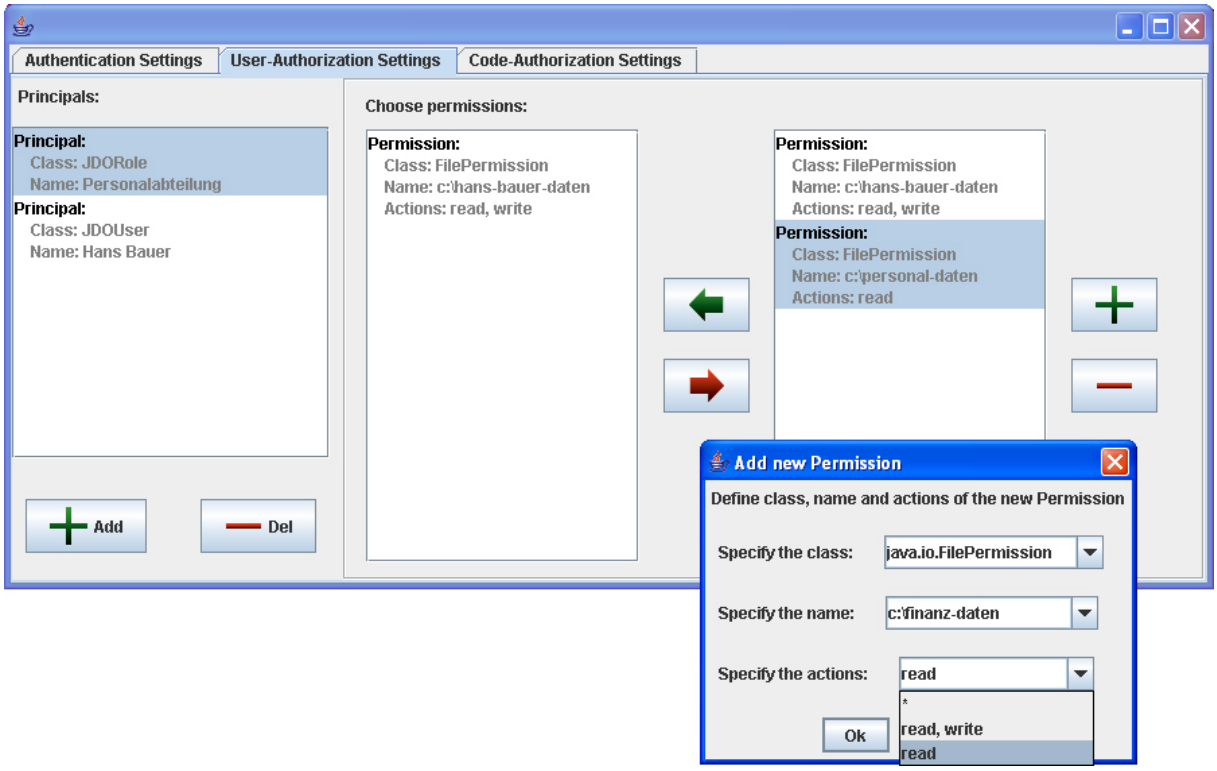

Abbildung 7: Zuweisung von Rechten zu Benutzer-Rollen, im RBAC-Verwaltungsmodus

5 Fazit und offene Punkte

Das in diesem Artikel vorgestellte Benutzer-, Rollen- und Rechte-Managementsystem erleichtert den Einsatz von JDOSecure. Die bislang getrennte Verwaltung von Benutzern/Rollen und Rechten in wenig komfortablen und z. T. unübersichtlichen Text-Dateien wurde durch eine Lösung ersetzt, bei der die Verwaltung komfortabel über eine Administrationsanwendung mit grafischer Benutzeroberfläche vorgenommen werden kann. Die zur Authentifizierung und Autorisierung notwendigen Daten werden hierbei in einer eigenständigen JDO-Ressource verwaltet.

Die Administrationsanwendung stellt derzeit zwei unterschiedlich mächtige Verwaltungsmodi zur Verfügung, mit deren Hilfe sich die Berechtigungen für Benutzer bzw. Rollen definieren lassen. Durch das flexible Datenmodell ist es möglich, Erweiterungen wie z. B. die Einführung eines Modus zur hierarchischen Verwaltung von Rollen, auf Ebene der Administrationsanwendung ohne Anpassungen am zugrundeliegenden Datenmodell vorzunehmen. Die Verwaltung der Daten in einer eigenständigen JDO-Ressource gestattet zudem den Einsatz des Benutzer-, Rollen- und Rechte-Managementsystems in nahezu jeder Umgebungen.

Als zukünftig Erweiterung ist die Vergabe und Kontrolle von Berechtigungen auf Objekt-Ebene vorgesehen. Dies erfordert allerdings die zusätzliche Erfassung von Objekt-Berechtigungen, ermöglicht jedoch auch die Realisierung von neue Anwendungsszenarien bei der Nutzung von JDOSecure.

Das vorgestellte Benutzer-, Rollen- und Rechte-Managementsystem vereinfacht aber bereits in der vorgestellten Version die Verwaltung von Benutzern, Rollen und Rechten erheblich und wird daher zu einer Steigerung der Akzeptanz von JDOSecure beitragen.

Literatur

Blosser, Jeremy (2000): *Explore the Dynamic Proxy API,* http://java.sun.com/developer/technicalArticles/ DataTypes/proxy/, 2000.

Ferraiolo, David F.; Sandhu, Ravi; Gavrila, Serban; Kuhn, D. Richard und Chandramouli, Ramaswamy (2001): *Proposed NIST Standard for Role-Based Access Control, ACM Transactions on Information and System Security*, 2001, *vol. 4 no. 3*, p. 224–274.

Gong, Li (2002): *Java 2 Platform Security Architecture,* http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-spec.doc.html, 2002.

Java Community Process (2004a): *JSR-012: Java Data Objects (JDO) Specification,* 2004.

Java Community Process (2004b): *JSR 59: J2SE Merlin Release Contents, J2SE 1.4, Maintenance Draft Review 5,* http://www.jcp.org/en/jsr/detail?id=59, 2004.

Merz, Matthias (2005): *JDOSecure: Ein Sicherheitsmodell für die Java Data Objects-Spezifikation,* Lehrstuhl für Wirtschaftsinformatik III, Universität Mannheim, Diskussionspapier 3-05, 2005.

Oaks, Scott (2001): *Java Security* The Java Series, Second, Sebastopol, CA, USA, O'Reilly & Associates, Inc., 2001.

Open Software Foundation (1995): *RFC 86.0 : Unified Login with Pluggable Authentication Modules (PAM),* 1995, http://www.opengroup.org/tech/rfc/mirror-rfc/rfc86.0.txt.

Samar, Vipin und Lai, Charlie (1996): *Making Login Services Independent of Authentication Technologies,* http://www.sun.com/software/solaris/pam/pam.external.pdf, 1996.

Sun Microsystems (2001): *Java Authentication and Authorization Service (JAAS), Reference Guide for the Java 2 SDK, Standard Edition, v 1.4,* http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/JAASRefGuide.html, 2001.

Sun Microsystems (2002): *JAAS Login Configuration File,* http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/tutorials/LoginConfigFile.html, 2002.

Sun Microsystems (2005): *The Java Language Specification*, 3rd, Addison-Wesley Professional, 2005.