Die datenbankzentrierte Web-Entwicklungsplattform Apex erfreut sich steigender Beliebtheit. Dabei wird sie nicht nur zur Neuentwicklung, sondern auch zur Erweiterung bestehender Applikationen eingesetzt. Eine häufige Anforderung in diesem Zusammenhang ist das Prinzip der Einmal-Anmeldung: Hat sich der Nutzer an der Hauptanwendung authentisiert, soll eine separate Anmeldung an der Apex-Anwendung entfallen. Dafür wird eine mögliche Lösung vorgestellt.

SSO zwischen Eclipse RCP und Apex: Vertrauen auf den ersten Blick

Anton Schlegel, PROMATIS software GmbH

Im vorliegenden Kapitel stellt der Horus Business Modeler, ein Werkzeug zur Geschäftsprozess-Modellierung (GPM), die zu erweiternde Lösung dar. Es nutzt eine Oracle-Datenbank und basiert auf der Eclipse Rich Client Platform (RCP). Das Werkzeug soll mithilfe von Apex um „Governance, Risk and Compliance"-Funktionalitäten (GRC) erweitert werden. Eine wichtige Anforderung ist dabei das Single-Sign-on (SSO). Das bedeutet, dass sich der Nutzer nach einer einmaligen Anmeldung am GPM-Werkzeug nicht zusätzlich an der Apex-Anwendung authentisieren muss. Nach einer Erläuterung der technischen Grundlagen werden die Funktionsweise der Lösung sowie die Methoden vorgestellt, die bei der Implementierung zum Einsatz kommen. Es folgt eine detaillierte Beschreibung der technischen Umsetzung.

Technische Grundlagen

Eclipse bietet mit RCP eine modulare und erweiterbare Plattform zur Entwicklung von Desktop-Anwendungen an. Die entwickelten Anwendungen sind von der Eclipse Integrated Development Environment (IDE) unabhängig. Um SSO zwischen Eclipse RCP und Apex umzusetzen, ist es wichtig zu verstehen, wie der Zugang zu den Apex-Applikationen geregelt ist.

Den Zugang zu einer Apex-Applikation regelt ein sogenanntes „Authentifizierungsschema". Dieses kann unter anderem auf Datenbank-Accounts oder auf Oracle-SSO-Servern basieren. Die erste Möglichkeit hat zur Folge, dass der Entwickler eine/mehrere Tabelle(n) mit Benutzernamen/Passwörtern pflegen muss. Die zweite erfordert eine Zusatzkomponente in Form eines Oracle-SSO-Servers. Bei der vorgestellten Lösung kommt dagegen eine dritte Variante zum Einsatz: ein benutzerdefiniertes Authentifizierungsschema. Dieses muss eine PL/SQL-Funktion (Authentifizierungsfunktion) enthalten, die die Logik dafür implementiert, wem der Zugang zur Anwendung gewährt wird und wem nicht. Der Artikel zeigt, wie die zu implementierende Logik ausgestaltet werden kann beziehungsweise welche weiteren Schritte in Apex und Drittanwendungen nötig sind.

Konzept

Wie in Abbildung 1 zu erkennen ist, wird Apex von der Eclipse-Anwendung heraus mithilfe einer URL aufgerufen. Diese URL beinhaltet Parameter, die der Nutzer-Identifikation dienen. Daraufhin erfolgt eine Umleitung zur Login-Seite. Ein Seitenprozess auf der Login-Seite extrahiert die Parameter aus der URL und übergibt sie an die Authentifizierungsfunktion. Diese soll sowohl die Benutzer des GPM-Werkzeugs als auch diejenigen, die die

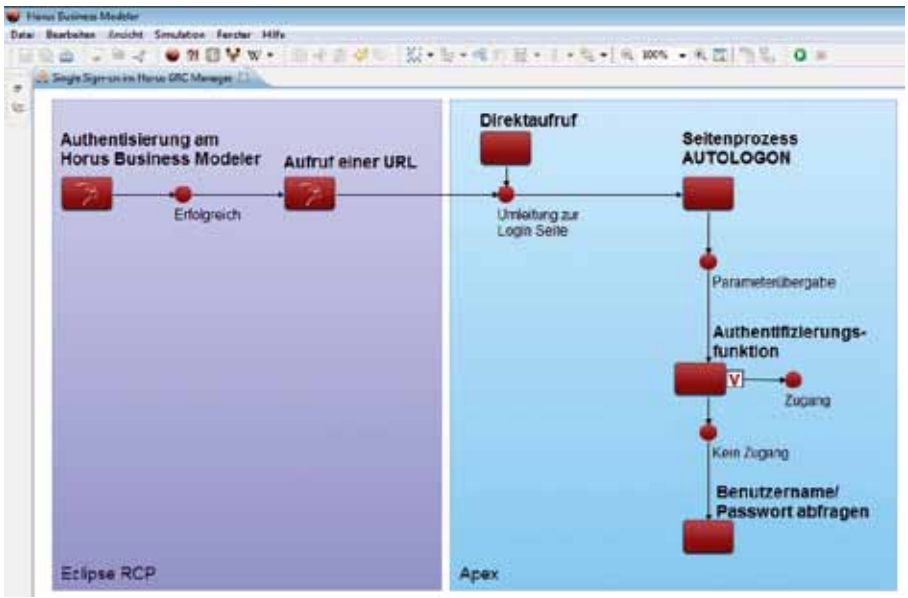

Abbildung 1: SSO-Konzeption

Apex-Anwendung direkt aufrufen, authentifizieren können.

Technische Umsetzung

Wie bereits erwähnt, werden an die Apex-Anwendung bestimmte Parameter in einer URL zur Identifikation des Nutzers übergeben. Hierbei handelt es sich zum einen um den Login-Namen des Benutzers und zum anderen um einen Hash-Wert beziehungsweise eine Prüfsumme.

Der Datensatz, der „gehasht" wird, besteht aus dem Login-Namen, einer Sitzungs-ID und einem Zeitstempel. Die Sitzungs-ID identifiziert eindeutig einen Nutzer des GPM-Tools, der aktuell am Werkzeug angemeldet ist. Der Zeitstempel sorgt dafür, dass die Prüfsumme nach Ablauf einer definierten Zeit nicht mehr gültig ist. Die Anmeldung an der Apex-Komponente ist ohne Eingabe von Benutzername/Passwort somit nur innerhalb eines bestimmten Zeitfensters möglich.

Die Bildung der Prüfsumme führt die Eclipse-Anwendung durch. Eine mögliche Ausgestaltung lässt sich in diesem Pseudocode beschreiben: „md5_hash_function(username, session_id, timestamp)". Eine URL, mit der jede Apex-Anwendung aufgerufen wird, ist so aufgebaut: „https://<Host>:<Port>/apex/f?p=<App>:<Seite>::<Request>:::<Parameterbezeichnung>:<Parameterwert>". Konkret könnte ein Aufruf in der Eclipse-Anwendung so aussehen: „https://horus-grc-manager:8080/apex/f?p=24313:4::AUTOLOGON::: P101_UNAME,P101_UPASS:tester1,2BC5617650F41F4A969DB4D8871256D2".

Darüber hinaus erfordert die Umsetzung des Konzepts eine benutzerdefinierte Authentifizierungsfunktion. Diese bildet einen Hashwert nach dem gleichen Muster wie die Eclipse-Anwendung, wofür sich die Prozedur „DBMS_OBFUSCATION_TOOLKIT.MD5" eignet. Diese setzt folgende Logik um: „Liefere dann „true" zurück, wenn der erzeugte Hash-Wert mit dem übergebenen Hash-Wert übereinstimmt oder wenn die Kombination Benutzername/Passwort von Apex als gültig verifiziert wird". Die Validierung des Passworts wird durch den Aufruf der Prozedur „APEX_UTIL.IS_LOGIN_PASSWORD_VALID" umgesetzt.

```
DECLARE
v_mixed VARCHAR2(4000);
v_user VARCHAR2(4000);
v_pass VARCHAR2(4000);
v_fsp_after_login_url VARCHAR2(4000) := :FSP_AFTER_LOGIN_URL;
BEGIN
SELECT SUBSTR(v_fsp_after_login_url, instr(v_fsp_after_login_
url,'P101_UNAME,P101_UPASS')+
LENGTH(,P101_UNAME,P101_UPASS')+1)
INTO v_mixed
FROM dual;
SELECT SUBSTR(v_mixed, 1, instr(v_mixed, ,,')-1) INTO v_user FROM
dual;
SELECT SUBSTR(v_mixed, instr(v_mixed, ,,')+1) INTO v_pass FROM
dual;
wwv_flow_custom_auth_std.login(
P_UNAME => v_user,
P_PASSWORD => v_pass,
P_SESSION_ID => v(,APP_SESSION'),
P_FLOW_PAGE => :APP_ID||':1'
);
END;
```

Listing 1

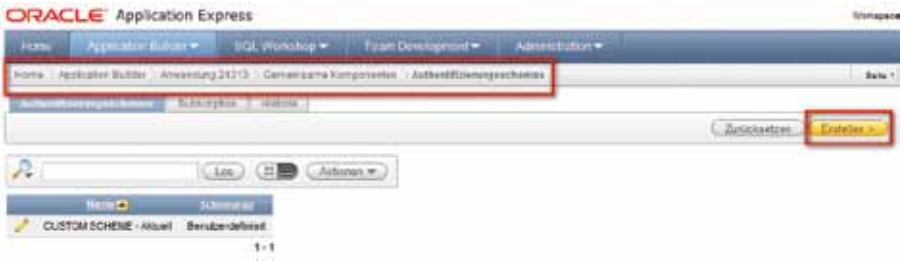

Abbildung 2: Authentifizierungsschema erstellen

In einem weiteren Schritt ist ein Authentifizierungsschema zu erstellen, das die Funktion nutzt. Das Vorgehen unterscheidet sich je nach Apex-Version geringfügig. Abbildung 2 bezieht sich auf die Version 4.1.

Das erstellte Authentifizierungsschema ist als aktuell zu markieren, damit es von der Applikation genutzt wird. Nach seiner Erstellung wird die Login-Seite der Apex-Anwendung angepasst. Auf dieser Seite ist ein Prozess „autologon" zu erstellen. Es muss sichergestellt sein, dass der Seitenprozess auf der Login-Seite als erstes ausgeführt wird, damit der Nutzer die Login-Seite nicht sieht (vorausgesetzt die Parameter in der URL sind valide). Der Seitenprozess soll „Vor Header" und mit der niedrigsten Sequenz ausgeführt werden. Zu beachten ist, dass dieser Prozess nur ausgeführt wird, falls die Anforderung (siehe „<Request>" in der URL oben) dem Ausdruck „AUTOLOGON" entspricht (siehe Abbildung 3).

So wird beim Aufruf der oben erwähnten URL der Prozess „autologon" ausgeführt. Da die Ausführung jedoch bereits vor dem Header-Aufbau stattfindet, sind die Elemente „P101_UNAME" und „P101_UPASS", die entsprechend den Benutzernamen und die Prüfsumme beinhalten, noch nicht gesetzt. Daher müssen die Werte erst aus der Apex-Anwendungsvariablen „FSP_AFTER_LOGIN_URL" mithilfe der Funktionen „instr" und „substr" extrahiert werden. Die extrahierten Werte übernimmt dann die Prozedur „wwv_flow_custom_auth_std.login", die dann im Hintergrund die Authentifizierungsfunktion aufruft (siehe Listing 1).

Abschließend ist zu erwähnen, dass als Übertragungsprotokoll „HTTP over SSL" zum Einsatz kommt. Sonst könnte ein potenzieller Angreifer die

Abbildung 3: Bedingungen für die Prozess-Ausführung

Parameter abfangen und sich ohne Kenntnis des Passworts mit dem Log-in-Namen und dem Hash-Wert an der Apex-Anwendung authentisieren. Eine verschlüsselte Übertragung verhindert derartige Angriffe.

Fazit

Mit der vorliegenden Lösung wurde ein einfacher Mechanismus zur Ein-mal-Anmeldung zwischen Anwendungen basierend auf Eclipse RCP und Apex-Applikationen aufgezeigt. Dabei ist weder die Pflege zusätzlicher Tabellen noch eine weitere Komponente wie der Oracle-SSO-Server notwendig. Ebenso wurde die Sicherheit insofern berücksichtigt, als die Anwendungen über eine verschlüsselte Verbindung miteinander kommunizieren. Die vorgestellte Vorgehensweise ist nicht auf den Einsatz mit Eclipse RCP beschränkt und lässt sich somit auch auf weitere Projekte übertragen.

Anton Schlegel
PROMATIS software GmbH
anton.schlegel@promatis.de