Vertrauen ist gut

WS-Trust: Ein Einstieg in die Praxis

Michel Alessandrini, Willi Nüßer, Michael Pollmeier

Mit Hilfe von WS-Security können diverse Sicherheitsanforderungen wie Integrität, Vertraulichkeit, Unabstreitbarkeit und Authentifizierung in einer Webservice-Architektur umgesetzt werden. Für die Verschlüsselung der Nachricht oder das Bilden einer digitalen Signatur sind dabei Sicherheits-Token erforderlich, zum Beispiel in Form von X.509-Zertifikaten. Diese müssen in reinen WS-Security-Landschaften manuell erstellt und an die Akteure verteilt werden. Wenn beispielsweise das Zertifikat eines Service-Providers ausläuft und erneuert wird, muss es von Hand an alle Service-Consumer verteilt werden. Dieses Konzept ist weder flexibel noch kostengünstig, sodass ein Automatismus hilfreich wäre. An dieser Stelle setzt WS-Trust an und erweitert die bestehende WS-Security-Spezifikation.

WS-Trust befindet sich derzeit in der „public review"-Phase und liegt in Version 1.3 vor [NGG06]. Planmäßig wird OASIS die vollständige Spezifikation Mitte 2007 abschließen. Teile der aktuellen Spezifikation werden jedoch bereits von einigen Frameworks prototypisch implementiert. Für Java-basierte Services wird derzeit Apache Rahas entwickelt, es wurde jedoch noch keine offizielle Version freigegeben, ebenso fehlen zum aktuellen Zeitpunkt noch Testfälle und Dokumentation.

Zertifikate in Webservice-Umgebungen

Abbildung 1 zeigt eine typische Situation bei der Absicherung einer Kommunikation zwischen Services. Der Client (*Credit Request*) benötigt die Zertifikate zweier Service-Provider (*ValidationService**) für den Zugriff, liegt aber nur mit einem der Services in der gleichen Vertrauensdomäne (d. h. z. B. gleiche Certification Authority, CA). Wenn sich die Zertifikate der Service-Provider ändern, sind aufwändige manuelle Eingriffe zur Verteilung der Zertifikate notwendig. Dies muss insbesondere über die Grenze der eigenen Vertrauensdomäne hinweg geschehen. Ein mühsames und anfälliges Verfahren.

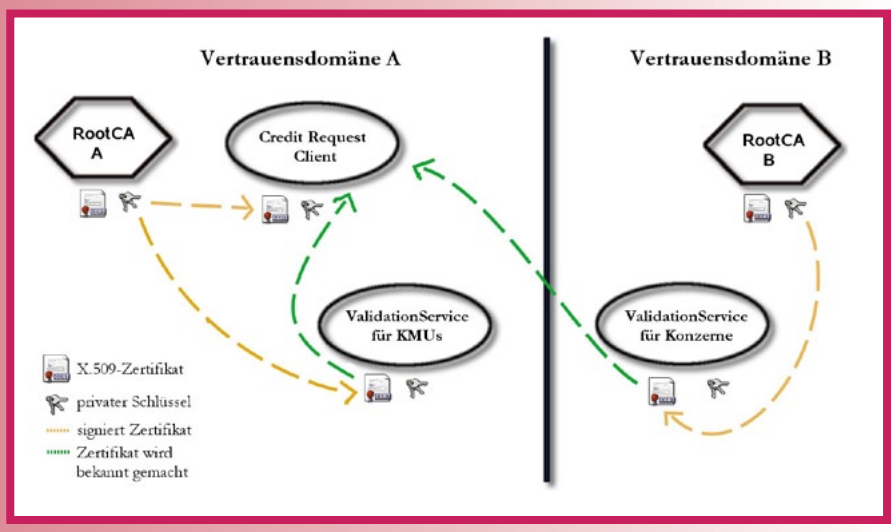

Abb. 1: Verteilung der Zertifikate ohne WS-Trust

WS-Trust

An dieser Stelle setzt der Standard WS-Trust an. Mittelpunkt der Spezifikation ist ein *Security Token Service (STS)*. Mit diesem können einige der in [ANP06] beschriebenen Probleme von WS-Security gelöst werden:

- *Sicherheits-Token-Verteilung:* Die Sicherheits-Token müssen nicht mehr manuell verteilt werden, sondern können zur Laufzeit beim STS angefordert werden. Bevor beispielsweise ein Service-Consumer seine Anfrage an den Service sendet, benötigt er dessen Zertifikat, um die Nachricht mit WS-Security zu verschlüsseln. Der STS ist in der Lage, Zertifikate zur Laufzeit auf Anfrage bereitzustellen. Er kann auch ein neues Zertifikat ausstellen, wenn das alte eines Service in Kürze abläuft. Die Service-Consumer müssen über das neue Zertifikat nicht separat informiert werden, da sie es vor einer Service-Anfrage automatisch beim STS abfragen.
- *Sicherheits-Token-Format:* WS-Security ist bewusst flexibel gehalten, sodass viele verschiedene Formate für Sicherheits-Token verwendet werden können. Dazu gehören beispielsweise X.509-Zertifikate, Kerberos-Tickets, SAML-Assertions und Benutzername/Kennwort-Kombinationen [OWS04]. Diese Flexibilität kann jedoch zu Problemen führen, wenn zwei Teilnehmer miteinander kommunizieren möchten, die das Token-Format des jeweils anderen nicht interpretieren können. An dieser Stelle kann der STS helfen, indem er zwischen den verschiedenen Formaten konvertiert.
- *Vertrauensverhältnis herstellen:* Wenn die Kommunikationspartner sich nicht gegenseitig vertrauen, kann dennoch ein Vertrauensverhältnis über den STS hergestellt werden. Er kann eine SAML-Assertion ausstellen, durch die der STS sein Vertrauen in den Akteur ausspricht. Mit diesem können die Kommunikationspartner untereinander ein Vertrauensverhältnis aufbauen.
- *Namensräume:* In einer heterogenen Architektur ist es möglich, dass die Kommunikationsteilnehmer semantisch dasselbe meinen, es syntaktisch aber unterschiedlich ausdrücken. Ein Teilnehmer könnte beispielsweise eine SAML-Assertion erhalten, die ihm versichert, dass sein Kommunikationspartner den Rang des „chief technical officer" besitzt. Wenn er aber nur die Bezeichnung „CTO" kennt, wird er die Assertion unter Umständen nicht korrekt einordnen. Beide Ränge haben die gleiche semantische Bedeutung, werden syntaktisch aber unterschiedlich ausgedrückt. Der STS kann in einer solchen Situation dafür genutzt werden, die Syntax zu vereinheitlichen, sodass beide Teilnehmer miteinander kommunizieren können. Auf diese Weise können mit WS-Trust heterogene Systeme miteinander verbunden werden. Der STS agiert als zentrale Vermittlungsinstanz, die zwischen den Systemen und Teilnehmern übersetzt.

Die Entwertung (Revocation) von Sicherheits-Token ist dagegen explizit *kein* Ziel der Spezifikation.

Wenn all diese Möglichkeiten, die sich durch eine WS-Trust-Architektur ergeben, genutzt werden, kann ein solcher STS sehr

komplex werden. Je nach bestehender Infrastruktur kann auch die Integration bestehender Sicherheitsdienste notwendig sein. Als Resultat verleiht WS-Trust einer serviceorientierten Architektur jedoch deutlich mehr Flexibilität. Die Komplexität der einzelnen Akteure kann deutlich reduziert werden. Außerdem können administrative Aufgaben wie das Verteilen der Sicherheits-Token automatisiert werden.

WS-Trust-Operationen

Ein STS bietet also verschiedene Dienste an. Alle Anfragen an den STS werden dabei in ein `RequestSecurityToken`-Element (`RST`) eingebettet. Dieser antwortet mit einer `RequestSecurityToken-Response (RSTResponse)`. Für die Umsetzung der oben genannten Funktionen sind lediglich drei Basisoperationen erforderlich:

- *Austausch von Token:* Der Anfragende liefert in der RST ein Sicherheits-Token, zum Beispiel ein X.509-Zertifikat. Durch eine digitale Signatur authentifiziert er sich beim STS und fordert ein neues Token an, beispielsweise eine SAML-Assertion. Auf diese Weise können Sicherheits-Token in verschiedene Formate konvertiert werden.
- *Ausstellung von Token:* Der Anfragende fordert eine Erneuerung seines bestehenden Sicherheits-Token an, beispielsweise weil es bald ungültig wird. Im engeren Sinne ist diese Operation lediglich ein Spezialfall der ersten Funktion.
- *Validierung von Token:* Ein Webservice wurde aufgerufen, kann den mitgelieferten Sicherheits-Token jedoch nicht validieren, zum Beispiel weil er dessen Format nicht interpretieren kann oder weil kein Vertrauensverhältnis zum Service-Consumer besteht. Diese Aufgabe kann jedoch an den STS delegiert werden.

Windows Communication Foundation

Microsoft ist der Java-Community mit seiner WS-Trust-Implementierung derzeit einen Schritt voraus. Das vormals unter dem Entwicklungsnamen *Indigo* bekannte Framework *Windows Communication Foundation (WCF)* vereint eine Reihe von Modulen für serviceorientierte Anwendungsentwicklung, die bislang getrennt waren [MWC06].

Zusätzlich sind neue Konzepte in das Framework eingeflossen, zum Beispiel eine neue Abstraktionsschicht für die Implementierung. Dadurch werden technische Details und Hintergründe verdeckt, wodurch sich sowohl Vor- als auch Nachteile ergeben. Es ist beispielsweise schwieriger nachzuvollziehen, welche Standards tatsächlich verwendet werden, da die Konfigurationssprache dies nicht immer hergibt. An dieser Stelle ist der Programmierer auf die Dokumentation von Microsoft angewiesen. Andererseits muss der Entwickler nicht mehr alle Spezifikationen und Anwendungsfälle im Detail kennen. Auf diese Weise wird die Komplexität reduziert und die Verbreitung der Standards gefördert.

WCF implementiert bereits einen Teil der aktuellen WS-Trust-Spezifikation. Es ist Teil von Microsofts .NET 3.0 und wird dadurch schnell eine hohe Verbreitung erfahren. In der Entwicklergemeinde stößt der Prototyp bereits auf Interesse und Unterstützung. So hat Pablo Cibraro (aka Cibrax) eine prototypische STS-Implementierung für WCF veröffentlicht, die als Basis für die im Folgenden vorgestellte Fallstudie dient [CWS06].

Fallstudie

Inhaltlich integriert sich die Fallstudie, die im Rahmen des EU-Forschungsprojektes SENSORIA [SEU05] umgesetzt wurde, in den Geschäftsablauf einer Bank, bei der ein Unternehmen einen Kredit beantragt. Ein Teil dieses Prozesses ist die Validierung der Unternehmensbilanz, um dessen Bonität zu prüfen. Der Service-Consumer (*Credit Request*) bestimmt anhand der Bilanzdaten, ob es sich um einen Konzern oder ein kleines bzw. mittleres Unternehmen (KMU) handelt. Anschließend delegiert der Credit Request Client die Bilanzprüfung an den jeweils zuständigen *ValidationService* (s. Abb. 2).

Zertifikate im Einsatz

Bevor der Credit Request Client einen Service unter Verwendung von WS-Security aufrufen kann, benötigt er das Zertifikat des Ziel-Service. Bei diesem Vorgang kommt der STS zum Einsatz, der die Verteilung der Zertifikate an zentraler Stelle übernimmt. Infolgedessen kann der Credit Request Client zunächst eine Anfrage an den STS stellen, um das aktuelle Zertifikat des Ziel-Service (*ValidationService*) zu erhalten. In dieser RST-Anfrage (`RequestSecurityToken`) authentifiziert sich der Client mit seiner digitalen Signatur, z. B. mittels X.509-Zertifikat. Der STS liefert daraufhin in der RST-Response eine SAML-Assertion, mit der sich der Client bei dem *ValidationService* authentifizieren kann (s. Abb. 2). Auf diese Weise muss die Verteilung der Zertifikate nicht mehr im Vorhinein manuell durchgeführt werden, sondern sie erfolgt zur Laufzeit. Letztendlich müssen so nicht mehr alle Clients bei einer Service-Zertifikatsänderung angepasst werden.

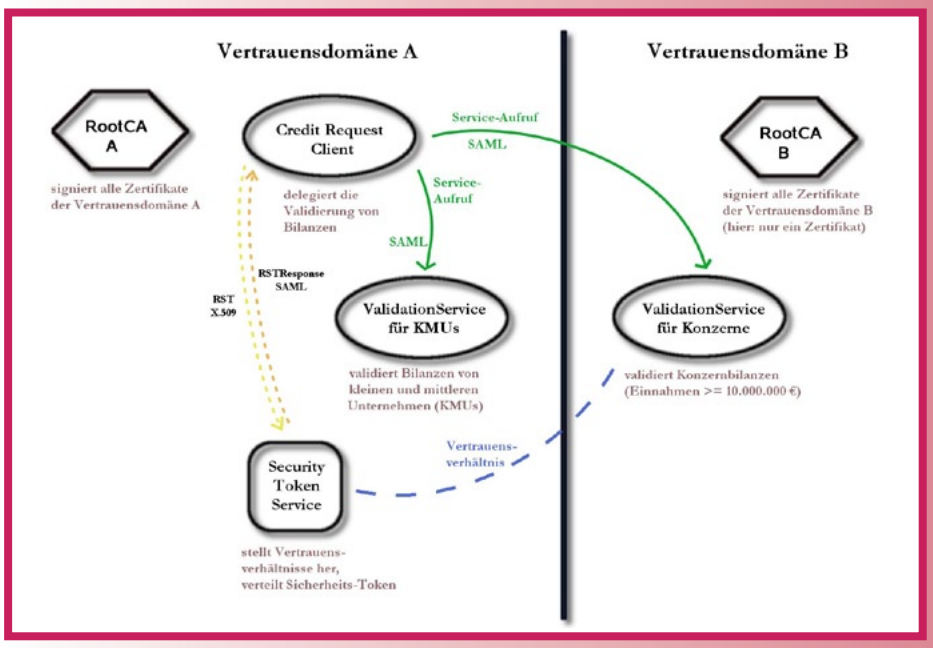

Abb. 2: Architekturskizze eines WS-Trust-Szenarios

Die Struktur der Authentifizierung vereinfacht darüber hinaus den Umgang mit Sicherheits-Token für die Ziel-Services. Diese müssen lediglich den offenen SAML-Standard interpretieren. Nur der STS muss mit allen verwendeten Client-Token umgehen können. Dafür kommen unter anderem X.509-Zertifikate, Kerberos-Tickets oder Benutzername/Kennwort-Kombinationen in Frage. Die Komplexität auf Seiten der Services wird auf diese Weise deutlich reduziert.

Verteiltes Vertrauen

Wie in Abbildung 2 zu sehen ist, befinden sich nicht alle Akteure in derselben Vertrauensdomäne. Es gibt zwei Zertifizierungsstellen, die alle Zertifikate innerhalb ihrer Domäne ausstellen, in der jeweils anderen aber unbekannt sind. Wenn der Credit Request Client (Domäne A) den *ValidationService für Konzerne* (Domäne B) nutzen möchte, muss also auf anderem Weg ein Vertrauensverhältnis hergestellt werden.

Auch hierfür kann der SecurityTokenService (STS) genutzt werden, da der Service aus Domäne B dem STS vertraut. Mit der signierten SAML-Assertion, die der STS dem Credit Request Client ausstellt, kann sich der Client beim *ValidationService für Konzerne* ausweisen. Der STS agiert also – neben seiner Funktion als Zertifikateverteiler – auch als Brücke und Vermittler (Router) zwischen den Vertrauensdomänen.

Implementierung des Prototyps

Im Folgenden werden die Abläufe zur Implementierung mit der Windows Communication Foundation erläutert. Die vollständige, lauffähige Fallstudie befindet sich im Downloadbereich [SIGS] zu diesem Artikel. Wir beginnen mit der Darstellung des Clients.

Weil beide *ValidationServices* prinzipiell dieselbe Aufgabe verrichten, nämlich die Prüfung von Unternehmensbilanzen, implementieren sie auch dasselbe Interface. Für den Credit Request Client ändert sich somit nur der Endpoint, je nachdem welcher Service aufgerufen werden soll (s. Listing 1).

```
ChannelFactory<IValidationService> factory =
  new ChannelFactory<IValidationService>(endpoint);
IValidationService validationService = factory.CreateChannel();
```

Listing 1: Vorbereitung eines Service-Requests; Client: Program.cs

Mehr Code als in Listing 1 ist für einen Service-Aufruf nicht erforderlich, alle anderen Aktionen werden in Konfigurationsdateien definiert. Die Endpoints für die beiden Ziel-Services sind in der Client-Konfiguration beschrieben (`App.config`, Auszug s. Listing 2).

Der grundsätzliche Aufbau ist für beide Services identisch, sie unterscheiden sich für den Client lediglich durch die URL. Die `bindingConfiguration` (s. Listing 2) beschreibt die Kommunikation mit dem *ValidationService für KMUs*. Die Nachrichten werden vom Credit Request Client signiert und für den Service verschlüsselt (`messageProtectionOrder`). Für die Authentifizierung wird ein `IssuedToken` verwendet, in diesem Fall das SAML-Token. Im Element `issuer` wird definiert, dass der SecurityTokenService dieses Token ausstellen soll. Auf diese Weise kann der Credit Request Client ein Vertrauensverhältnis zu den beiden Services aufbauen.

```
<binding name="ServiceBinding">
  <security messageProtectionOrder="SignBeforeEncrypt"
            authenticationMode="IssuedToken">
    <issuedTokenParameters tokenType=
"http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1">
      <issuer address=
          "http://localhost/WCFSecurityTokenService/service.svc"
        bindingConfiguration="MutualCertificateBinding"
        binding="customBinding">
        <identity>
          <dns value="WCFSecurityTokenService"/>
        </identity>
      </issuer>
    </issuedTokenParameters>
  </security>
  <httpTransport/>
</binding>
```

Listing 2: Credit Request Client: ServiceBinding für die Ziel-Services (App.config)

Die Kommunikation des Clients mit dem STS ist natürlich ebenfalls mit WS-Security abgesichert. Das dafür erforderliche Zertifikat des SecurityTokenService extrahiert der Client aus seinem Windows-Keystore. Dies ist in der hier nicht aufgeführten `behaviorConfiguration` konfiguriert [SIGS].

Mit diesen Schritten ist der Credit Request Client so konfiguriert, dass er sich zunächst beim SecurityTokenService authentifiziert. Von diesem erhält er ein signiertes SAML-Token sowie das Zertifikat des gewünschten *ValidationService*. Mit diesen Informationen kann sich der Client beim Service authentifizieren und die Nachricht verschlüsseln.

SecurityTokenService

Für die Fallstudie konnte die Implementierung des STS vollständig aus der zugrunde liegenden Implementierung von Pablo Cibraro [CWS06] übernommen werden. An dieser Stelle werden nur die wichtigsten Abläufe skizziert.

```
[ServiceContract]
public interface ISecurityTokenService {
  [OperationContract(Action = RSTConstants.Trust.Actions.Issue,
    ReplyAction = RSTConstants.Trust.Actions.IssueReply)]
  Message IssueToken(Message rstMessage);
}
```

Listing 3: WCF.SAML.ISecurityTokenService

Die Schnittstelle zum STS ist in einem Interface definiert (s. Listing 3). Darin wird mit Hilfe von Konstanten festgelegt, dass die eingehenden RST-Nachrichten für die Ausstellung von neuen Token von der Methode `IssueToken()` behandelt werden.

In der Implementierung des Interface erfolgt zunächst eine formale und inhaltliche Prüfung des RST. Anschließend erstellt der STS einen `RSTResponse`, der eine SAML-Assertion sowie das Zertifikat des Ziel-Service enthält (s. Listing 4). Dazu wird zunächst der gewünschte Zielservice durch seine URL identifiziert. Die Zuordnung zwischen URL und entsprechendem Zertifikat ist in der Konfiguration des STS festgelegt, die in Ausschnitten in Listing 5 zu sehen ist.

Damit wird dann das gewünschte Zertifikat aus dem Windows-Keystore extrahiert (s. Listing 4, `encryptionToken`) und in die Antwort eingebettet. Der öffentliche Schlüssel ergibt sich aus dem Zertifikat (s. Listing 4, `x509Key`). Der vollständigen Code befindet sich in der Methode `GenerateSymmetricProofKey()` in der Datei `SamlSecureTokenService.cs` im Verzeichnis `Common` in [SIGS].

```
EndpointAddress appliesTo = rst.AppliesTo;
X509SecurityToken encryptionToken = (X509SecurityToken)
  SamlConfiguration.SamlTokenIssuerConfiguration.
  GetServiceToken(appliesTo);
X509AsymmetricSecurityKey x509Key =
  new X509AsymmetricSecurityKey(encryptionToken.Certificate);
```

Listing 4: Zertifikat des Ziel-Service aus dem lokalen Windows-Keystore

```
<WcfSaml>
  <samlTokenIssuer ttlInSeconds="300">
    <serviceTokens>
      <!-- SAML Authority certificate -->
      <add uri="http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Issue"
        storeLocation="LocalMachine" storeName="My"
        findValue="CN=WCFSecurityTokenService"
        findType="FindBySubjectDistinguishedName"/>
      <!-- services -->
      <add uri=
          "http://localhost/WCFValidationServiceCorporates/service.svc"
        storeLocation="LocalMachine" storeName="My"
        findValue="CN=WCFValidationServiceCorporates"
        findType="FindBySubjectDistinguishedName"/>
      <add uri="http://localhost/WCFValidationServiceKMUs/service.svc"
        storeLocation="LocalMachine" storeName="My"
        findValue="CN=WCFValidationServiceKMUs"
        findType="FindBySubjectDistinguishedName"/>
    </serviceTokens>
  </samlTokenIssuer>
</WcfSaml>
```

Listing 5: STS: Konfiguration der Zertifikate

Bewertung und Ausblick

WS-Trust erweitert die bestehende WS-Security-Spezifikation um viele nützliche Funktionen. Insbesondere mit der automatischen Verteilung der Sicherheits-Token können lästige administrative Tätigkeiten eingespart werden. Ein weiterer Vorteil ist die höhere Flexibilität, da verschiedene Token-Formate vom SecurityTokenService verarbeitet und in andere Formate konvertiert werden können. Dabei wird vorzugsweise die offene SAML-Spezifikation verwendet, welche die Interoperabilität zwischen heterogenen Systemen erleichtert.

Die Fallstudie zeigt den Nutzen einer WS-Trust-Architektur anhand der bereits zur Verfügung stehenden Funktionalität der Windows Communication Foundation (WCF). Sie demonstriert auch den Einsatz der STS als Brücke, um Vertrauensverhältnisse zwischen verschiedenen Domänen herzustellen. Auf Basis der bestehenden Implementierung sind weitere interessante Szenarien denkbar. So könnte beispielsweise eine Single-Sign-On-Lösung realisiert werden. Der Client müsste dazu lediglich die vom STS erzeugte SAML-Assertion – mit der er sich bei den Services authentifizieren kann – cachen.

Derzeit ist Microsoft mit seinem WCF ein Vorreiter für WS-Trust. Vergleichbare Frameworks für andere Webservice-Plattformen befinden sich noch in der Entwicklung. Im Apache-Rahas-Projekt entsteht zurzeit eine Lösung für Axis2, die bislang jedoch aufgrund mangelnder Dokumentation und Testfälle noch nicht praxistauglich ist. Mit der Veröffentlichung der finalen Version von WS-Trust Mitte 2007 ist jedoch ein Fortschritt in dieser Hinsicht zu erwarten. Eine zentrale Frage wird dann auch die Interoperabilität zwischen den verschiedenen Plattformen sein.

Literatur und Links

[ANP06] M. Alessandrini, W. Nüßer, M. Pollmeier, WS-Security in komplexen Landschaften in: JavaSPEKTRUM, 4/2007

[CWS06] P. M. Cibraro, SAML - STS implementation for WCF, ASP.NET-Entwicklergemeinde, September 2006, `http://weblogs.asp.net/cibrax/archive/2006/09/08/SAML-_2D00_-STS-implementation-for-WCF.aspx`

[MTR03] P. Madsen, WS-Trust: Interoperable Security for Web Services, xml.com (O'Reilly), 24 Juni 2003, `http://www.xml.com/pub/a/ws/2003/06/24/ws-trust.html?page=1`, Abruf: 11.12.06

[MWC06] Windows Communication Foundation, `http://wcf.netfx3.com`

[NGG06] Nadalin, Goodner et al., OASIS; WS-Trust 1.3 Committee Draft 01 (2006-09), `http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-rddl.html`, Abruf: 11.12.06

[OWS04] OASIS Web Services Security (WSS), `http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf`

[RSW06] M. McRae, FW: Public Review of WS-SX WS-SecureConversation v1.3 and WS-Trust v1.3, OASIS, tc-announce message, 15. September 2006, `http://lists.oasis-open.org/archives/tc-announce/200609/msg00008.html`, Abruf: 12.12.06

[RWT06] M. Raepple, WS-Security: Neue Standards für mehr Sicherheit (2006-05), in: iX, 5/2006, `http://www.heise.de/ix/artikel/2006/05/126/`, Abruf: 11.12.06

[SEU05] Sensoria, Software Engineering for Service-Oriented Overlay Computers, `http://www.sensoria-ist.eu`

[SIGS] vollständige, lauffähige Fallstudie zum Artikel, `http://www.sigs-datacom.de/sd/publications/js/2007/04/index.htm`

Michel Alessandrini ist an der FHDW als wissenschaftlicher Mitarbeiter mit den Forschungsschwerpunkten SOA und künstlicher Intelligenz tätig. Des Weiteren ist er für die S&N AG als Projektleiter in einem EU-Forschungsprojekt im Einsatz. E-Mail: michel.alessandrini@fhdw.de.

Dr. Willi Nüßer ist seit 2002 Professor für angewandte Informatik an der Fachhochschule der Wirtschaft (FHDW) in Paderborn und Inhaber der Heinz Nixdorf Stiftungsprofessur. Er war von 1996 bis 2002 bei der SAP AG u. a. im LinuxLab in Walldorf tätig. Seit 2002 beschäftigt er sich in Theorie und Praxis mit der Webservice-Technologie. E-Mail: wilhelm.nuesser@fhdw.de.

Michael Pollmeier ist Absolvent der FHDW. Er ist bei der S&N AG, Paderborn im Bereich des Designs, der Entwicklung und der Sicherung von Webservices tätig. E-Mail: pollmeier@gmx.de.

Weitere Informationsquellen

http://www.sigs-datacom.de/sd/publications/js/2007/04/index.htm