

# Projektowanie obiektowe oprogramowania

## Wykład 14

### Architektura systemów (1)

#### Interoperacyjność

## Wiktor Zychła 2014

---

### 1 Architektura systemów

**Aplikacje rozległe** (ang. *Enterprise applications*) – to wielomodułowe systemy informatyczne, często rozwijane przez lata lub powstające w wyniku połączenia kilku niezależnych elementów, zbudowanych w różnych technologiach i w oparciu o różne konstrukcje architektury.

Najprostszy przykład – połączenie systemów informatycznych dwóch (lub więcej) dużych banków. Inny przykład – zintegrowany miejski/gminny/powiatowy system informatyczny, obejmujący różne obszary odpowiedzialności podmiotu Zamawiającego.

W obszarze architektury systemy rozległe rodzą wyzwania **integracyjne**. Integracja z kolei oznacza

- przepływ informacji wewnątrz systemu – np. przepływ danych między modułami składowymi
- przepływ informacji o tożsamości użytkownika – np. pojedyncze logowanie

Implementacje mechanizmów integracyjnych powinny charakteryzować się właściwością **interoperacyjności**, czyli otwartości technologicznej (patrz dokumenty **Krajowe Ramy Interoperacyjności, European Interoperability Framework**).

### 2 WebServices

Usługi aplikacyjne są jednym z podstawowych narzędzi interoperacyjności. Usługa definiuje kontrakt operacyjny specyfikowany w formalnej postaci **WSDL** (Web Service Description Language) i protokół komunikacyjny (zwykle oparty o HTTP lub przynajmniej TCP). Językiem wymiany komunikatów jest XML/JSON. Wykorzystywana do budowy crossplatformowych rozwiązań komunikacyjnych.

#### 2.1 Demo

Brak dema, bo technologia jest oczywista

### 3 X.509/PKCS#12

Standard X.509 opisuje infrastrukturę PKI (**Public Key Infrastructure**), czyli usług kryptograficznych opartych o **cyfrowe certyfikaty**, umożliwiające bezpieczne szyfrowanie i podpisywanie danych. Standard PKCS#12 opisuje format pliku służący do przenoszenia cyfrowych certyfikatów (rozszerzenia plików to \*.p12 lub \*.pfx).

### 3.1 Demo

Podczas wykładu zobaczymy podstawowe pojęcia związane z PKI/PKCS#12.

Użyjemy narzędzia do generowania certyfikatów (Portecle), zaimportujemy certyfikat do systemowego zasobnika oraz nauczymy się korzystać z certyfikatu w zasobniku do podpisywania i szyfrowania danych.

```
// przeszukanie zasobnika
StoreName name          = StoreName.My;
StoreLocation location = StoreLocation.CurrentUser;

X509Store store = new X509Store( name, location );

store.Open( OpenFlags.ReadOnly );

X509Certificate2Collection certificates = store.Certificates;

X509Certificate2 certificate = null;
for ( int i = 0; i < certificates.Count; i++ )
{
    X509Certificate2 cert = certificates[i];

    // wydobycie certyfikatu
    if ( cert.SubjectName.Name.ToLower() == "foo" ||
        cert.FriendlyName.ToLower() == "foo"
        )
    {
        certificate = new X509Certificate2( cert );
        break;
    }
}
for ( int i = 0; i < certificates.Count; i++ )
{
    certificates[i].Reset();
}

if ( certificate != null )
{
    string thestring = "foobar";

    // podpisanie
    RSACryptoServiceProvider csp =
        (RSACryptoServiceProvider)certificate.PrivateKey;

    byte[] data = Encoding.UTF8.GetBytes( thestring );

    string oid    = CryptoConfig.MapNameToOID( "SHA1" );
    byte[] signed = csp.SignData( data, oid );
    string signeds = Convert.ToBase64String( signed );

    // weryfikacja podpisu
    csp = (RSACryptoServiceProvider)certificate.PublicKey.Key;
    if ( !csp.VerifyData( data, oid, signed ) )
        throw new CryptographicException();

    // szyfrowanie
    csp = (RSACryptoServiceProvider)certificate.PublicKey.Key;
    byte[] encrypted = csp.Encrypt( data, false );
}
```

```

string encrypteds = Convert.ToBase64String( encrypted );

// odszyfrowywanie
byte[] encrypted2 = Convert.FromBase64String( encrypteds );

csp = (RSACryptoServiceProvider)certificate.PrivateKey;
byte[] decrypted = csp.Decrypt( encrypted2, false );

string decrypteds = Encoding.UTF8.GetString( decrypted );
}

```

## 4 XMLDSig/XAdES

**XMLDsig** to opublikowany przez W3C standard podpisywania elektronicznych dokumentów XML. Jest bardziej ogólny niż PGP i inne standardy oparte na podpisywaniu danych binarnych – wśród zalet należy wymienić m.in. możliwość wielokrotnego podpisywania tego samego dokumentu (kontrasygnaty) czy fakt, że podpisany dokument pozostaje czytelny dla człowieka (sygnatura nie modyfikuje struktury podpisywanego dokumentu).

Do tworzenia sygnatur (podpisów) używane są algorytmy kryptografii asymetrycznej, stąd dobra współpraca ze standardem PKCS#12 (przenoszenie certyfikatów).

XMLDSig jest podstawą dla wielu interoperacyjnych standardów wymiany danych, w tym XAdES (patrz niżej) czy SAML (patrz dalej).

Przykład dokumentu źródłowego:

```

<?xml version="1.0" encoding="UTF-8"?>
<Osoba>
  <DaneOsobowe>
    <Nazwisko>Kowalski</Nazwisko>
    <Imie>Jan</Imie>
    <PESEL>12341234243</PESEL>
    <DataUrodzenia>1990-02-01T16:45:58.433+01:00</DataUrodzenia>
  </DaneOsobowe>
</Osoba>

```

Dokument podpisany:

```

<?xml version="1.0" encoding="UTF-8"?>
<Osoba>
  <DaneOsobowe>
    <Nazwisko>Kowalski</Nazwisko>
    <Imie>Jan</Imie>
    <PESEL>12341234243</PESEL>
    <DataUrodzenia>1990-02-01T16:45:58.433+01:00</DataUrodzenia>
  </DaneOsobowe>
  <Signature xmlns="http://www.w3.org/2000/09/xmlldsig#">
    <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmlldsig#rsa-sha1" />
      <Reference URI="">

```

```

    <Transforms>
      <Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
      <Transform Algorithm="http://www.w3.org/TR/2001/REC-xm1-c14n-
20010315" />
    </Transforms>
    <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
    <DigestValue>FqghDrPlyjNVDSyWZ80s8M7geaM=</DigestValue>
  </Reference>
</SignedInfo>
<SignatureValue>k14ZCnSBQoux...g==</SignatureValue>
<KeyInfo>
  <X509Data>
    <X509Certificate>MIICkTCCAXkCBFB8APAwDQYJKo...==</X509Certificate>
  </X509Data>
</KeyInfo>
</Signature>
</Osoba>

```

**XAdES** – zbiór rozszerzeń dla XMLDsig, dodający m.in. timestamp, czyli znakowanie podpisu w czasie (w taki sposób żeby wiadomo było certyfikat podpisujący dokument był ważny w momencie podpisywania dokumentu).

Standardu XAdES używa się w praktyce, patrz: np. System e-Deklaracje, specyfikacje wejścia/wyjścia:

[http://www.finance.mf.gov.pl/documents/766655/1196432/eDek\\_Specyfikacja\\_We-Wy\\_1.9.3\\_Test.pdf](http://www.finance.mf.gov.pl/documents/766655/1196432/eDek_Specyfikacja_We-Wy_1.9.3_Test.pdf)

Ten sam dokument co poprzednio podpisany jako XAdES. Proszę zwrócić uwagę na węzeł **SignedSignatureProperties** zawierający informację o czasie złożenia podpisu.

```

<Osoba>
  <DaneOsobowe>
    <Nazwisko>Kowalski</Nazwisko>
    <Imie>Jan</Imie>
    <PESEL>12341234243</PESEL>
    <DataUrodzenia>1990-02-01T16:45:58.433+01:00</DataUrodzenia>
  </DaneOsobowe>
  <ds:Signature Id="SignatureId" xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xm1-c14n-
20010315" />
      <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
      <ds:Reference URI="">
        <ds:Transforms>
          <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xm1-c14n-20010315"
/>
          <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature" />
        </ds:Transforms>
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <ds:DigestValue>JXqGse6B1AdtDDr8MdREuju/2CA=</ds:DigestValue>
      </ds:Reference>
      <ds:Reference URI="#SignedPropertiesId"
Type="http://uri.etsi.org/01903/v1.1.1#SignedProperties">
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <ds:DigestValue>jjiZAD9KVuR1G9NJ6qdXbCNnpZY=</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue Id="SignatureValueId">Ha0g/nZaA51W0GWGRrpJ9GBzXyK//f...
  </ds:SignatureValue>
  <ds:KeyInfo>
    <ds:X509Data>
      <ds:X509Certificate>MIIDRCCAq2gA... </ds:X509Certificate>
    </ds:X509Data>

```

```

        </ds:KeyInfo>
        <ds:Object Id="XadesObject">
          <QualifyingProperties Target="#SignatureId"
xmlns="http://uri.etsi.org/01903/v1.1.1#">
            <SignedProperties Id="SignedPropertiesId">
              <SignedSignatureProperties>
                <SigningTime>2014-06-02T23:18:01</SigningTime>
                <SigningCertificate>
                  <Cert>
                    <CertDigest>
                      <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
                        <DigestValue>fc9oOIPntILmorvUDMKX+TYBW/s</DigestValue>
                      </CertDigest>
                      <IssuerSerial>
                        <ds:X509IssuerName>DC=qux, DC=bar, DC=foo,
SERIALNUMBER=PESEL:123456789, CN=Subject123</ds:X509IssuerName>
                      <ds:X509SerialNumber>00A34ECBFBE85C5EE0541BEC4723D227</ds:X509SerialNumber>
                    </IssuerSerial>
                  </Cert>
                </SigningCertificate>
                <SignaturePolicyIdentifier>
                  <SignaturePolicyImplied />
                </SignaturePolicyIdentifier>
              </SignedSignatureProperties>
            </SignedProperties>
          </QualifyingProperties>
        </ds:Object>
      </ds:Signature>
    </Osoba>

```

## 4.1 Demo

Podpisanie dokumentu XML na podstawie „Interoperable XML Digital Signatures”

<http://www.wiktorzychla.com/2012/12/interoperable-xml-digital-signatures-c.html>

[http://www.wiktorzychla.com/2012/12/interoperable-xml-digital-signatures-c\\_20.html](http://www.wiktorzychla.com/2012/12/interoperable-xml-digital-signatures-c_20.html)

[http://www.wiktorzychla.com/2012/12/interoperable-xml-digital-signatures-c\\_4247.html](http://www.wiktorzychla.com/2012/12/interoperable-xml-digital-signatures-c_4247.html)

XaDES

[http://www.microsoft.com/france/openness/open-source/interoperabilite\\_xades.aspx](http://www.microsoft.com/france/openness/open-source/interoperabilite_xades.aspx)

## 5 Single sign-on

**Single sign-on** (pojedyncze logowanie) – to właściwość aplikacji rozległych, w których dostęp do tych części poszczególnych modułów które wymagają autentykacji i autoryzacji, możliwy jest po jednokrotnym potwierdzeniu tożsamości użytkownika.

Z uwagi na różne implementacje realizujące ten sam cel, można mówić o wzorcu dla aplikacji rozległych.

Najprostsza, na co dzień spotykana implementacja SSO wbudowana jest w systemy operacyjne – po jednokrotnym zalogowaniu dostaje się dostęp do aplikacji, które o tożsamość użytkownika odpytują system operacyjny. Takie SSO nie jest interesujące, ciekawie robi się dopiero wtedy, kiedy mówimy o SSO poza granicami jednego systemu – na przykład kiedy interfejs użytkownika osadzony jest w przeglądarce internetowej i dostaje się on do różnych witryn, rozsianych gdzieś po świecie.

Istnieją różne możliwości implementacji tego wzorca. Jednym z ważniejszych kryteriów właściwego wyboru jest zgodność ze standardami przemysłowymi.

Wśród powszechnie akceptowanych protokołów SSO należy wymienić:

- OpenID – dobry wybór ale phishing + brak single sign off
- OAuth2
- CAS (Central Authentication Service)
- SAML-p
- Shibboleth
- **WS-Federation** – Office365, Sharepoint 2010, Windows 8, Azure Cloud Services

## 6 Claims-based authentication

**Claim** (stwierdzenie/oświadczenie) – informacja o **Kimś** wydane przez jakiegoś **Wystawcę**. Claim powinien być „podpisany” tzn. nie powinno być wątpliwości że wydał go **Wystawca**.

Zwykle nie da się nijak inaczej stwierdzić czy claim jest prawdziwy czy nie, ale **ufamy** wystawcy wobec czego **akceptujemy** informację.

Przykład: Stwierdzenie – „Jan Kowalski urodził się 04.11.1978”.



Jest to oświadczenie z „podpisem”, powszechnie akceptowane w bankach, sklepach itd. Fakt akceptowania wynika z relacji zaufania do Wystawcy oświadczenia.

## 7 WS-Federation

### 7.1 Pojęcia

Protokół WS-Federation przenosi pojęcia „oświadczenia” i „wystawcy” na język techniczny:

**Security Token Service (STS)** – wystawca oświadczeń, posiada informacje o użytkownikach aplikacji rozległej lub zna lokalizacje innych wystawców

**Oświadczenie** – czwórka (Type, Issuer, Subject, Value)

**Security Assertion Markup Language (SAML)** – dialekt XML zapisu oświadczeń, standaryzujący m.in. ich podpisywanie cyfrowe (XMLDsig). SAML mówi tylko tym jak skonstruowane są tokeny. Nie mówi o tym jak je wymieniać (język vs protokół). Na SAML opartych jest kilka różnych protokołów: WS-Federation, Google SSO, Shibboleth, SAML-protocol)

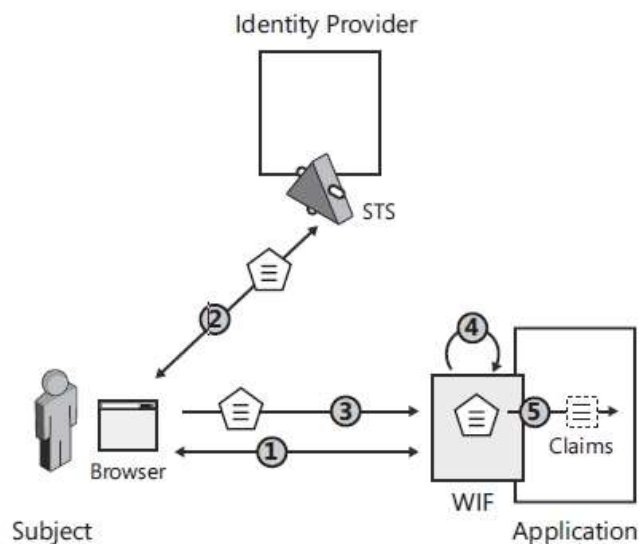
**Typowe oświadczenia** – nazwa użytkownika, imię, nazwisko, e-mail, adres, role (uprawnienia)

**Security token** (token bezpieczeństwa) – zbiór oświadczeń

**Relying Party (RP)**– aplikacja która ufa claimom wydanym przez STS

Mówi się „RP jest sfederowany (*federated*) z STS” = RP ufa oświadczeniom wydanym przez STS. W praktyce jest to równoważne stwierdzeniu „RP akceptuje zbiór czwórek (Type, Issuer, Subject, Value) podpisany znanym mu certyfikatem STSa”.

## 7.2 Protokół pojedynczego logowania



1. Użytkownik próbuje uzyskać dostęp do części systemu rozległego wymagającej autoryzacji. Aplikacja wymusza przekierowanie sesji przeglądarki do aplikacji – wystawcy oświadczeń
2. Wystawca oświadczeń weryfikuje tożsamość użytkownika (lub wykorzystuje fakt że tożsamość była już sprawdzana wcześniej), tworzy podpisany token SAML i przekazuje go do przeglądarki
3. Przeglądarka przekazuje otrzymany token do aplikacji wymagającej autoryzacji
4. Aplikacja wykorzystuje technologię umożliwiającą przetwarzanie oświadczeń (tu: Windows Identity Foundation) (lub przetwarza oświadczenia samodzielnie) w tym weryfikuje poprawność ich podpisu
5. Zestaw oświadczeń jest dostępny dla aplikacji

## 7.3 Protokół pojedynczego wylogowywania

1. Wystawca oświadczeń śledzi żądania wydania tokenu bezpieczeństwa – magazynuje adresy aplikacji występujących o oświadczenia

2. Po otrzymaniu żądania *wylogowania*, wystawca generuje do przeglądarki zasób (stronę) zawierającą adresy wszystkich aplikacji, które dotychczas w imieniu użytkownika wystąpiły o token bezpieczeństwa, ale dodaje do tych adresów parametr oznaczający wylogowanie (tu: *wsignoutcleanup1.0*).
3. Przeglądarka kieruje żądania do wszystkich kolejnych aplikacji
4. Aplikacje wykonują sobie tylko znaną procedurę wylogowania użytkownika z sesji

## 7.4 Bezpieczeństwo protokołu

Bezpieczeństwo protokołu WS-Federation oparte jest o 4 certyfikaty X509 (wszystkie poza jednym są opcjonalne):

- (O) Certyfikat bezpiecznych połączeń do serwera aplikacji (SSL)
- (O) Certyfikat bezpiecznych połączeń do serwera wystawcy oświadczeń (SSL)
- Podpisanie oświadczeń przez wystawcę oświadczeń (podpisany SAML)
- (O) Szyfrowanie wystawianych oświadczeń certyfikatem aplikacji

## 7.5 Inne cechy protokołu

Relacja zaufania do wystawcy jest przechodnia – jeżeli klient (RP) prosi o oświadczenia, a wystawca (STS) przekieruje jego żądanie do kolejnego wystawcy (a ten z kolei dalej itd.) to w efekcie ostateczny zbiór oświadczeń może być sumą oświadczeń wydanych przez kolejnych wystawców, a klient w ogóle nie musi być świadomy tego przez ile „węzłów” wystawców przeszło żądanie.

To daje możliwość budowania „bramek” (gateway), które na zewnątrz (dla klienta) implementują protokoły WS-Federation, a wewnątrz pozyskują oświadczenia albo od innego wystawcy WS-Federation albo z usługi logowania innego protokołu.

## 7.6 Demo

Obejrzymy szczegółowo przebieg sekwencji zdarzeń protokołu dla przykładowych RP/STS. Omówimy elementy charakterystyczne implementacji. W zależności od tego ile zostanie nam czasu, zbudujemy na żywo implementację klienta (RP) i/lub serwera (STS).

Obejrzymy scenariusze federacji przechodniej – gdzie system STS świadczy usługę na rzecz klienta, ale sam uwierzytelnia się w innym STS.

## 8 Literatura

Patterns & Practices – „A Guide to Claims-based Identity and Access Control” (darmowy ebook), <http://msdn.microsoft.com/en-us/library/ff423674.aspx>

Vito Bertocci – “Programming Windows Identity Foundation”