

# Projektowanie obiektowe oprogramowania

## Wykład 14

### Architektura systemów (1)

### Interoperacyjność

Wiktor Zychła 2024

---

## Spis treści

1	Architektura systemów .....	3
2	Domain-Driven Design.....	4
2.1	Strategic Design.....	4
2.2	Tactical Design.....	4
2.3	DDD or not DDD?.....	6
3	Service-Oriented Architecture .....	7
4	X.509/PKCS#12 .....	8
4.1	Pojęcia .....	8
5	XMLDSig/XAdES .....	10
5.1	Materiały do studiowania .....	12
6	Single sign-on.....	13
6.1	Informacje ogólne .....	13
6.2	Claims-based authentication.....	13
6.3	Federalizacja protokołów logowania.....	14
7	WS-Federation.....	15
7.1	Pojęcia .....	15
7.2	Protokół pojedynczego logowania .....	15
7.3	Protokół pojedynczego wylogowywania.....	16
7.4	Bezpieczeństwo protokołu.....	16
8	Interoperacyjność przemysłowa – przykład ePUAP .....	17
8.1	ePUAP .....	17
8.2	Ustawa węzłowa – Węzeł Krajowy.....	17
9	Literatura .....	18



# 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/ogólnokrajowy system informatyczny, obejmujący różne obszary funkcjonalne.

Jeszcze inny przykład – ekosystem usług w ramach chmury jednego dostawcy (Microsoft, Google).

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

- przepływ **informacji** wewnątrz systemu – np. przepływ danych między modułami składowymi
- przepływ **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 Domain-Driven Design

**Domain-Driven Design** to **podejście** do projektowania oprogramowania, w którym nacisk kładzie się na wierne odwzorowanie domen (dziedzin) problematyki biznesowej.

- DDD **nie jest** metodyką wytwarzania oprogramowania
- DDD **nie jest** typem architektury
- DDD **jest** zbiorem praktyk i wskazówek ułatwiających projektowanie systemów

### 2.1 Strategic Design

Etap projektowania

- rozpoznanie istoty zagadnienia, odpowiedzi na pytania „co”, „gdzie”?
- identyfikacja **domen**
  - domeny główne – realizują podstawowe cele oprogramowania
  - domeny pomocnicze – realizują dodatkowe cele oprogramowania
  - domeny generyczne – zagadnienia techniczne, np. zarządzanie tożsamością (logowanie, uprawnienia)
- identyfikacja pojęć w ramach domen (model pojęciowy)

**Bounded Context** wyznacza granice w których model pojęciowy jest spójny w ramach pojedynczej domeny. W tych granicach uczestnicy projektu posługują się jednym spójnym językiem określającym pojęcia. Ten język to **Ubiquitous Language**<sup>1</sup>.

W różnych kontekstach to samo pojęcie może oznaczać co innego, przykładowe pojęcia o możliwym różnym znaczeniu: jednostka, poziom, oddział, podmiot itp.

### 2.2 Tactical Design

Etap wytwarzania

- zejście na poziom techniczny, odpowiedzi na pytania „jak”?

Pojęcia modelu dziedzinowego są klasyfikowane według następującego podziału:

**Entity** – byt samodzielny, jednoznacznie identyfikowany przez **unikalny identyfikator**, którym można posługiwać się nawet poza granicami domeny. Wszystkie atrybuty są modyfikowalne, ponieważ równość jest równoważna równości identyfikatorów.

```
{  
  "id": "1234-4567",  
  "name": "Jan"
```

---

<sup>1</sup> W naszej dotychczasowej narracji w ramach wykładu byłby to po prostu **model pojęciowy** – czyli zbiór pojęć i relacji między nimi

```
    "surname": "Kowalski"
}
```

**Value Object** – byt niesamodzielny, nie ma identyfikatora, jego właściwości są niemodyfikowalne; dwa obiekty są równe jeśli wszystkie ich atrybuty są równe

```
{
  "city": "Wrocław",
  "street": "Kowalska"
  "number": "1"
}
```

**Entity** może być obiektem złożonym, zawierającym bądź to inne **Entity** bądź jakieś **Value Objects**.

Taki złożony obiekt nazywa się **Aggregate**, a o encji która znajduje się w korzeniu agregatu powiemy że jest to **Aggregate Root**.

```
{
  "id": "1234-4567",
  "name": "Jan"
  "surname": "Kowalski"
  "address" : {
    "city": "Wrocław",
    "street": "Kowalska"
    "number": "1"
  }
}
```

**Aggregate Root** jest zwykle obiektem transakcyjnym, to znaczy że zmiana jego stanu jest odwzorowywana w **Repozytorium**. **Aggregate Root** na poziomie kodu jest modelowany jako klasa, która zawiera logikę biznesową:

- walidacja
- składowanie
- inne procesy związane ze stanem lub zachowaniem

**Service** – usługa (technicznie: klasa) w ramach domeny, która związana jest z logiką zachowania ale nie posiada żadnego stanu i nie jest związana bezpośrednio z konkretnym Entity ani Value Object.

**Usługi w warstwie domeny** to coś innego niż **usługi w warstwie infrastruktury**<sup>2</sup>. Przykładem usługi w warstwie domeny jest usługa transferu środków między dwoma kontami (w teorii mogłaby to być metoda obiekty konta ale ponieważ dotyczy dwóch kont, można ją wynieść do usługi).

Jeśli chodzi o współpracujące z podejściem DDD rozwiązania dotyczące architektury, to sprawdzą się

- **architektura heksagonalna** – omówione już przez nas podejście służące lepszej izolacji warstwy logiki biznesowej
- **microservices** – architektura w której granice logiczne kontekstów stają się również granicami fizycznymi

---

<sup>2</sup> W naszej dotychczasowej narracji **usługi w warstwie infrastruktury** to po prostu porty wtórne (np. wysyłanie wiadomości email)

W oryginalnym opracowaniu Evansa, szczególnie w obszarze **Tactical Design**, pojawiają się pomocnicze pomysły:

- **Context Map** – wydzielona warstwa translacji między różnymi kontekstami
- **Shared Kernel** – część modelu współdzielona między kontekstami
- **Anticorruption Layer** – oddzielenie zewnętrznych systemów, schowanie ich za jawnymi warstwami tłumaczącymi (adaptery, fasady)
- **Separate Ways** – nie należy „na siłę” utrzymywać w ramach jednego kontekstu takiej jego części która jest kłopotliwa, czasem lepiej wydzielić osobny kontekst
- Itd.

To raczej zbiory wskazówek i porady niż nieodłączne elementy DDD

## 2.3 DDD or not DDD?

### Zalety

- DDD może pomóc zorganizować projektowanie i wytwarzanie aplikacji o dobrze określonej dziedzinie biznesowej, szczególnie w sytuacji kiedy problematyka jest nowa dla zespołu

### Wady

- Istnieją zagadnienia które nie modelują się naturalnie w ten sposób, szczególnie zagadnienia bardziej techniczne (np. jądro systemu operacyjnego)
- Abstrahowanie obiektów dziedziny od szczegółów składowania może powodować konieczność posiadania dodatkowych warstw w implementacjach repozytoriów. To może skutkować mniejszą wydajnością

### 3 Service-Oriented Architecture

**Service-Oriented Architecture** – pojęcie odnoszące się do architektury typu klient-serwer w której kontraktem między klientem a serwerem są operacje biznesowe.

Usługi aplikacyjne są jednym z podstawowych narzędzi interoperacyjności. Usługa zwykle definiuje kontrakt operacyjny specyfikowany w formalnej postaci, m.in.

- WSDL ([Web Service Description Language](#))
- WADL ([Web Application Description Language](#))
- [OpenAPI](#)

i protokół komunikacyjny (zwykle oparty o http). Językiem wymiany komunikatów jest XML/JSON.

Różnica między myśleniem typu SOA a myśleniem typu zwykła usługa typu REST jest dość subtelna i naszym zdaniem sprowadza się do kontraktów usług. Czyli technicznie usługa typu SOA może być implementowana jako usługa REST ale na poziomie kontraktu jest bardziej złożona i na poziomie wejścia/wyjścia operuje **agregatami** (patrz poprzednie rozdziały) a nie atomowymi obiektami.

Przykład.

Usługi REST z punktami końcowymi

/user/add  
/address/add  
/permission/add

i klient który chce dodać użytkownika z adresem i kilkoma uprawnieniami – musi wiedzieć w jakiej kolejności i które usługi wywoływać żeby uzyskać efekt.

Usługa SOA z punktem końcowym

/user/add

i klient który chce dodać użytkownika z adresem i kilkoma uprawnieniami – do punktu końcowego wysyła cały agregat opisujący użytkownika (z adresem i uprawnieniami) a punkt końcowy realizuje całą usługę (dodanie użytkownika, sprawdzenie czy trzeba dodać adres i czy trzeba dodać uprawnienia).

## 4 X.509/PKCS#12

Standard X.509 opisuje infrastrukturę PKI (**P**ublic **K**ey **I**nfrastructure), 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).

### 4.1 Pojęcia

**X.509** – standard przemysłowy infrastruktury przechowywania i przenoszenia [certyfikatów klucza publicznego](#). Stosowane przemysłowo certyfikaty są zwykle oparte na algorytmach [RSA](#) lub [ECC](#).

**PKCS#12** – standard formatu pliku umożliwiającego przenoszenie wielu par certyfikat – klucz prywatny

**Szyfrowanie za pomocą certyfikatu** – użycie klucza publicznego certyfikatu do szyfrowania danych. Tylko posiadacz klucza prywatnego może deszyfrować dane.

**Podpisywanie za pomocą certyfikatu** – użycie klucza prywatnego certyfikatu do szyfrowania danych. Każdy posiadacz klucza prywatnego może sprawdzić poprawność szyfrowania. W praktyce stosuje się podpisywanie nie samych danych, a **skrót**, wyliczonego za pomocą jakiejś funkcji skrótu (np. SHA256). Takie podejście daje gwarancję niedużego rozmiaru podpisu, niezależnego od rozmiaru wyjściowych danych.

Do generowania certyfikatu można użyć narzędzia (Portecle/Keystore Explorer/OpenSSL). W niektórych systemach operacyjnych certyfikat można importować do systemowego zasobnika.

Przykładowy kod używający certyfikatu do szyfrowania podpisywania.

```
// przeszukanie zasobnika certyfikatów
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 );
}

```

## 5 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/xmlldsig#enveloped-signature" />
          <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmlldsig#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-xml-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-xml-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:Signature>
  <ds:KeyInfo>
    <ds:X509Data>
      <ds:X509Certificate>MIIDRDCCAq2gA...</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>
            </Cert>
            <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>

```

## 5.1 Materiały do studiowania

Podpisanie dokumentu XML na podstawie „Interoperable XML Digital Signatures” (C#/Java)

<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)

Biblioteka XaDES (C#)

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

## 6 Single sign-on

### 6.1 Informacje ogólne

**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 Connect / OAuth2
- CAS (Central Authentication Service)
- SAML2.0
- Shibboleth
- **WS-Federation** – Office365, Sharepoint 2010, Windows 8, Azure Cloud Services

### 6.2 Claims-based authentication

Protokoły SSO do zwracania informacji o użytkowniku często używają tzw. oświadczeń (claims).

**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.

Podobnie w przypadku protokołu SSO, informacja zwrotna z serwera może zawierać informacje o użytkowniku takie jak imię, nazwisko, login, email czy PESEL, a aplikacja ufa dostawcy tożsamości – przyjmuje więc że przedstawione w przepływie logowania dane są poprawne (zweryfikowane).

### 6.3 Federalizacja protokołów logowania

Relacja zaufania do wystawcy jest **przechodnia** – jeżeli aplikacja prosi o oświadczenia, a dostawca tożsamości 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 jakiś protokół SSO, a same pozyskują oświadczenia albo od innego dostawcy (wszystko odbywa się przez przekierowania przez przeglądarkę).

**Przykład:** użytkownik inicjuje sesję do aplikacji A. Aplikacja A przekierowuje na dostawcę tożsamości DT. Tam, na ekranie logowania użytkownik widzi opcję „zaloguj za pomocą kolejnego dostawcy tożsamości” (KDT).

Po kliknięciu i kolejnym przekierowaniu, użytkownik ostatecznie loguje się do KDT.

Usługa KDT przez przeglądarkę zwraca informacje o użytkowniku do DT. Tam informacje są weryfikowane i/lub korelowane z własną bazą DT i kolejny zbiór informacji o użytkowniku jest zwracany do A.

A dostaje informację o sesji użytkownika uwierzytelnionej przez DT. A nie musi nawet wiedzieć, że ostatecznie użytkownik logował się za pomocą KDT.

## 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)

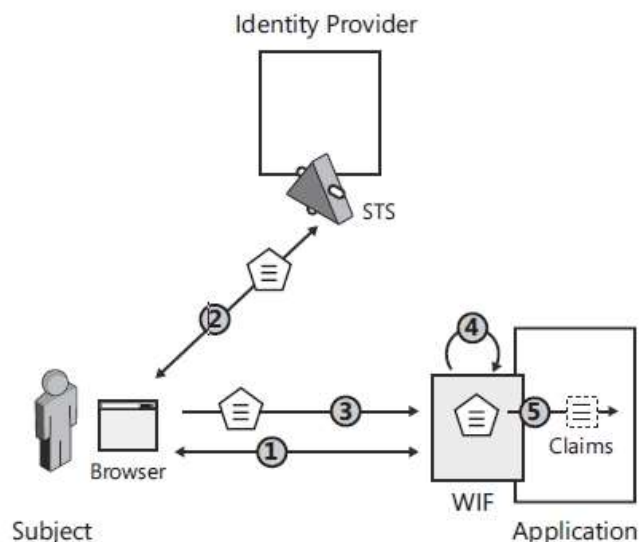
**Typowe oświadczenia** – nazwa użytkownika, imię, nazwisko, e-mail, adres, role (uprawnienia)

**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):

- (Opt) Certyfikat bezpiecznych połączeń do serwera aplikacji (SSL) (uniemożliwia podsłuchanie komunikacji)
- (Opt) Certyfikat bezpiecznych połączeń do serwera wystawcy oświadczeń (SSL) (uniemożliwia podsłuchanie komunikacji)
- Podpisanie oświadczeń przez wystawcę oświadczeń (podpisany SAML) (uniemożliwia sfałszowanie tokena)
- (Opt) Szyfrowanie wystawianych oświadczeń certyfikatem aplikacji (uniemożliwia wykorzystanie tokena wydanego aplikacji do uwierzytelnienia się w innej aplikacji)



## 8 Interoperacyjność przemysłowa – przykład ePUAP

### 8.1 ePUAP

Omówione wcześniej narzędzia interoperacyjne obejrzymy na przykładzie implementacji – platformy ePUAP (Elektroniczna Platforma Usług Administracji Publicznej) (<https://epuap.gov.pl>).

Platforma udostępnia szereg usług administracyjnych dla obywateli, w tym np. składanie wniosków o wydanie dowodu osobistego, bez potrzeby osobistej wizyty w urzędzie.

W obszarze autentykacji – platforma ePUAP wykorzystuje uwierzytelnianie federacyjne za pomocą protokołu SAML2, dodatkowo implementuje mechanizm delegowania uwierzytelniania (logowanie za pomocą dostawcy, np. banku). Jest to bardzo poprawne i bardzo praktyczne.

W obszarze integracji danych – platforma ePUAP udostępnia dla aplikacji szereg usług, m.in. usługę podpisywania dokumentów oraz obsługi tzw. elektronicznych skrzynek podawczych. Usługi te są zaimplementowane w interoperacyjnym standardzie WS-Security.

### 8.2 Ustawa węzłowa – Węzeł Krajowy

W lipcu 2018 weszła w życie [Ustawa o zmianie ustawy o usługach zaufania oraz identyfikacji elektronicznej oraz niektórych innych ustaw](#) (Dz.U. 2018 poz. 1544), która powołuje do życia tzw. Węzeł Krajowy na potrzeby uwierzytelniania użytkowników do e-usług publicznych oraz niepublicznych.

Usługa jest już dostępna pod adresem <https://login.gov.pl> i w kolejnych latach kolejne usługi będą stopniowo integrowane z Węzłem Krajowym.

Technicznie – węzeł krajowy jest podobnie jak ePUAP dostawcą usługi SSO w standardzie SAML2, od implementacji ePUAP różni się dość istotnymi szczegółami (m.in. wymaga certyfikatów ECDA a nie RSA).

## 9 Literatura

E. Evans – [“Domain-Driven Design: Tackling Complexity in the Heart of Software”](#)

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

Understanding WS-Security <https://msdn.microsoft.com/en-us/library/ms977327.aspx>

SAML2.0 [https://en.wikipedia.org/wiki/SAML\\_2.0](https://en.wikipedia.org/wiki/SAML_2.0) <http://saml.xml.org/saml-specifications>

ePUAP – dokumentacja dla integratora <https://epuap.gov.pl> i dalej Strefa urzędnika / Pomoc / Dla integratorów

Węzeł krajowy – dokumentacja dla integratora <https://mc.bip.gov.pl/interoperacyjnosc-mc/wezle-krajowy-dokumentacja-dotyczaca-integracji-z-wezlem-krajowym.html>

OldMusicBox.EIH.Client – przykładowa biblioteka kliencka do integracji z Węzłem Krajowym  
<https://github.com/wzychla/OldMusicBox.EIH.Client>