OPC UA

Príklady v C#

Abstrakt

Dokument bol vypracovaný ako súčasť predmetu Metódy výpočtovej inteligencie.

Marián Hók

2017

Obsah

1 Úvod

V dnešnej dobe je pojem „Industry 4.0“ veľmi populárny. Z histórie poznáme tri veľké priemyselné revolúcie: mechanizácia výroby pomocou vody (parné stroje), druhá predstavila sériovú výrobu pomocou elektrickej energie a tretia bola digitalizácia. Industry 4.0 predstavuje štvrtú etapu priemyselnej revolúcie – inteligentné továrne alebo tiež Industry Internet Of Things.

V továrňach budúcnosti budú všetky komponenty ovládať výrobný proces autonómne, vďaka ich prepojiteľnosti. Komponenty budú zbierať a zdieľať informácie a zároveň budú automaticky oznamovať ak potrebujú servis. Vzájomné prepojenie je teda najdôležitejšou časťou štvrtej priemyselnej revolúcie a podstatnou súčasťou tohto procesu je štandard OPC UA, ktorý zaručuje interoperabilitu na všetkých úrovniach výroby.

2 OPC UA

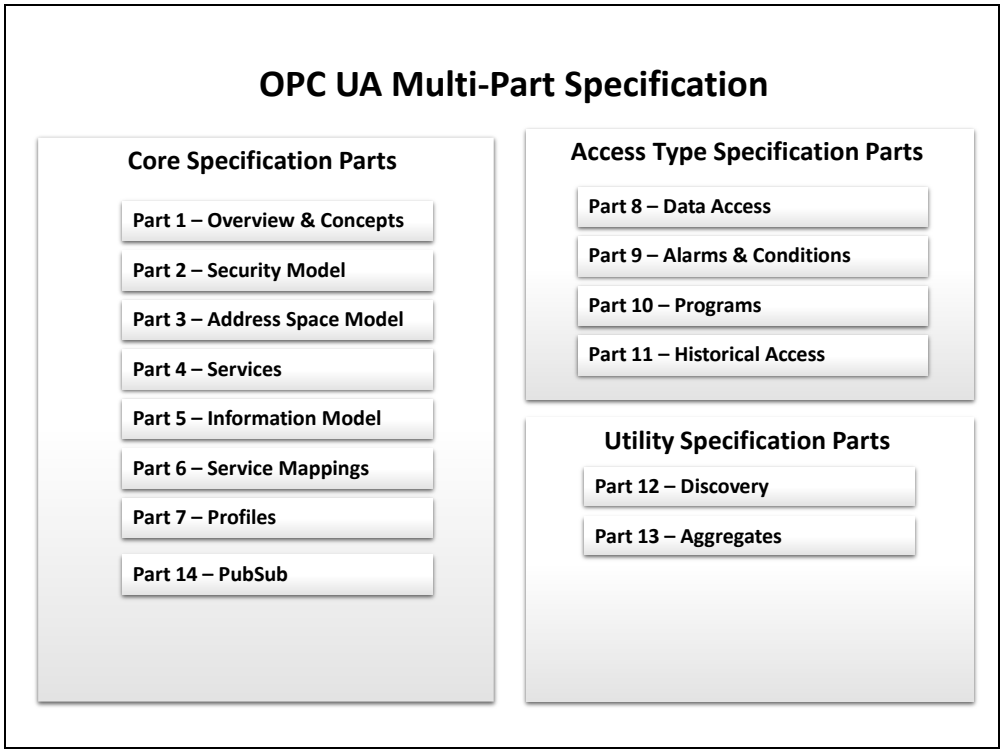
OPC - Open Connectivity je momentálne najviac rozšírený štandardizovaný proces výmeny dát pre automatizačnú technológiu. Umožňuje zber a prenos dát v jednotnej podobe a to z rôznych zariadení, riadiacich systémov a aplikácií v celej organizácii. Dizajn tohto štandardu umožňuje mapovať takmer všetky priemyselné dáta do dátovej štruktúry OPC.

OPC UA je vylepšená verzia štandardu OPC, ktorá má zjednotenú architektúru (unified architecture), vďaka čomu sa stáva platformovo nezávislým protokolom. Okrem toho má zabudované bezpečnostné mechanizmy a aplikácie sú plne škálovateľné od mikrokontrolerov až po podnikové servery.

“OPC UA poskytuje metódu pre bezpečnú a spoľahlivú výmenu dát a vzhľadom na to, že je najpopulárnejším otvoreným štandardom konektivity, bude hrať hlavnú úlohu v tom, aby sa Industry 4.0 stal realitou.”

2.1 Štruktúra OPC UA

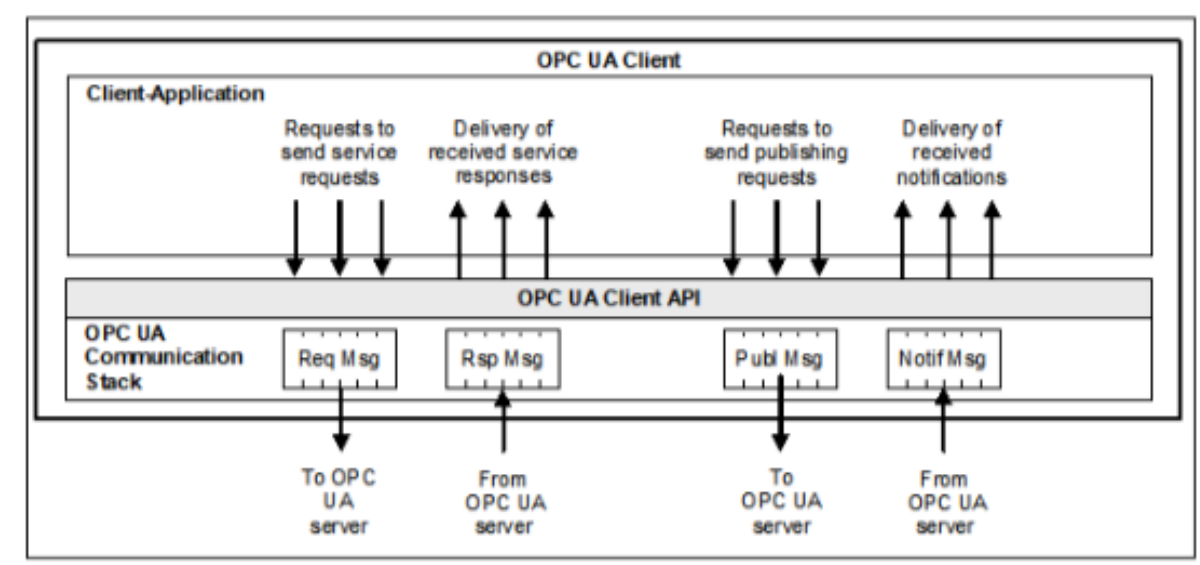
Časti od 1 do 7 s polu so 14. časťou špecifikujú základné schopnosti OPC UA. Definujú Adresný priestor a služby ktoré na ňom dokážu bežať. Časť 14 definuje vzor pre vydavateľov a odoberateľov (publisher, subsriber). Časti 8 až 11 špecifikujú prístup k dátam, alarmy a udalosti a prístup k minulým dátam. Časť 12 popisuje mechanizmy objavovania (hľadania serverov) a posledná 13. časť popisuje spôsoby agregácie dát.



Obrázok 1: Štruktúra špecifikácie OPC UA

2.2 Architektúra klienta

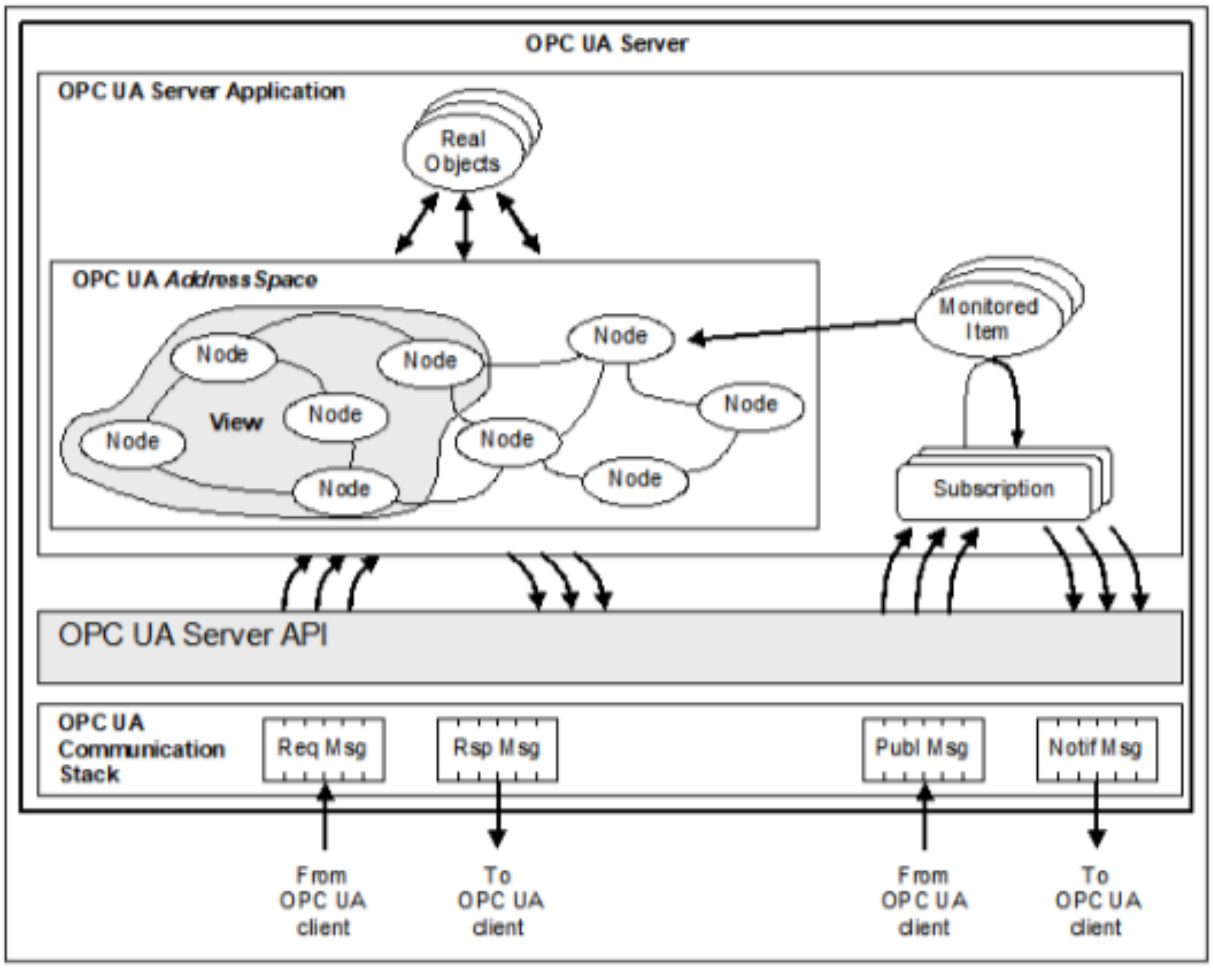
OPC UA klient sa skladá z dvoch častí: Klientská aplikácia a aplikačné rozhranie. Aplikačné rozhranie obsahuje komunikačnú vrstvu, ktorá dokáže komunikovať so serverom buď spôsobom požiadavka – odpoveď, alebo môže vytvoriť publish požiadavku, ktorá vytvorí na serveri subscription a server následne pravidelne upozorňuje klienta na nejakú udalosť. Aplikačné rozhranie klienta teda slúži na prepojenie klienta so serverom. Samotná klientská aplikácia už len žiada od API, aby vytvorilo nejakú požiadavku a spracúva dáta.



Obrázok 2: Architektúra OPC UA klienta

2.3 Architektúra servera

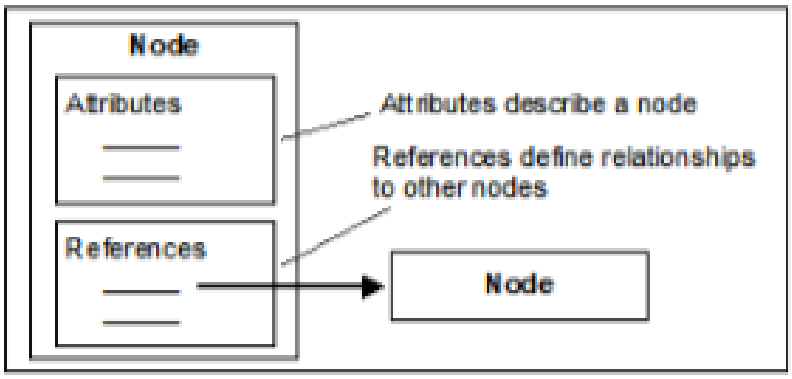
OPC UA Server sa podobne ako klient skladá zo serverovej aplikácie a serverového komunikačného API. Api prijíma požiadavky od klienta a odpovedá na ne. Serverová aplikácia však má tiež definovanú presnú štruktúru, obsahuje Adresný priestor, ktorý uchováva dáta a skladá sa z uzlov, ktoré sú vzájomne poprepájané a spolu tvoria objekty. V prípade, že klient zašle publish požiadavku, server u seba vytvorí subscribtion, ktorou monitoruje niektorý z uzlov a pravidelne zasiela informácie o tomto uzle. Reálne objekty komunikujú s adresným priestorom a sú to napríklad senzory alebo výrobné stroje.



Obrázok 3: Architektúra OPC UA servera

2.3.1 Model uzlov

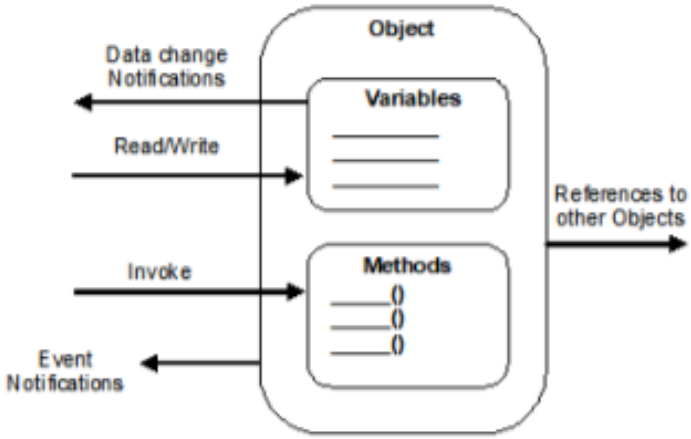
Uzly v adresnom priestore servera majú tiež presne definovanú štruktúru. Skladajú sa zo základných atribútov, medzi ktoré patria: NodeID, NodeClass, BrowseName, DisplayName, Description, WriteMask, UserWriteMask. Okrem atribútov obsahujú uzly aj referencie na iné uzly, čím sa definujú prepojenia medzi uzlami.



Obrázok 4: Model uzlu

2.3.2 Model objektov

Množiny uzlov spolu vytvárajú v adresnom priestore objekty. Každý objekt môže mať svoje atribúty a metódy a môže odkazovať na iné objekty. K atribútom objektu je možné pristupovať – čítať ich a zapisovať do nich, zároveň však atribúty vedia upozorňovať na svoju zmenu.



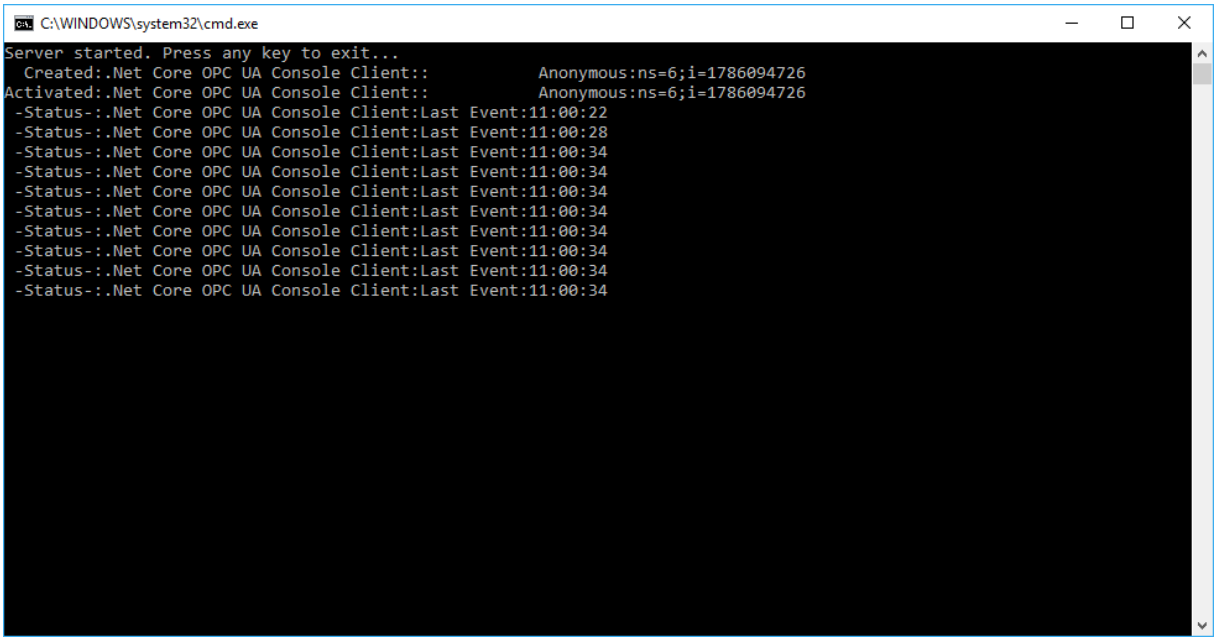
Obrázok 5: Model objektu v adresnom priestore

3 .NET Core Aplikácia

Aplikácie .NET Core sú multiplatformové a sú spustiteľné napríklad na operačných systémoch Windows, Linux, Mac. Testujeme vzorovú aplikáciu od OPCFoundation, ktorá je vytvorená pomocou knižnice OPCFoundation/UA-.NetStandardLibrary. Spustenie je možné cez príkazový riadok pomocou nástroja „dotnet“ alebo vo vývojárskom prostredí Visual Studio 2015 alebo 2017.

3.1 Server

Serverová konzolová aplikácia má vytvorený základný adresný priestor a umožňuje základnú komunikáciu medzi klientom serverom. Na obrázku môžeme vidieť, že server sa spustil, pripojil sa naň klient, ktorému bolo na základe platného certifikátu pripojenie umožnené. Server akonáhle client vytvorí na serveri reláciu, server začne odosielať informácie o poslednej udalosti, ktorú klient vyvolal. Po niekoľkých sekundách sme klienta vypli, klient teda nevykonáva žiadnu činnosť a teda sa nemení ani čas poslednej udalosti na serveri.



Obrázok 6: Ukážka spustenej serverovej aplikácie v .NET Core

S vytvorením servera nám pomáha pomocný projekt zo spomínanej knižnice – Opc.Ua.Server. Tento projekt obsahuje triedy, ktoré zjednodušujú vytvorenie servera, správu subcsripcií, diagnostiku, vytvorenie adresného priestoru a iné.

3.1.1 Tvorba serveru

```
ApplicationInstance.MessageDlg = new ApplicationMessageDlg();
ApplicationInstance application = new ApplicationInstance();

application.ApplicationName = "UA Core Sample Server";
application.ApplicationType = ApplicationType.Server;
application.ConfigSectionName = "Opc.Ua.SampleServer";

//načítanie serverovej konfigurácie.
ApplicationConfiguration config = await
application.LoadApplicationConfiguration(false);

//kontrola certifikátu.
bool haveAppCertificate = await application.CheckApplicationInstanceCertificate(false,
0);
```

```
   if (!haveAppCertificate)
     {
       throw new Exception("Application instance certificate invalid!");
     }

if (!config.SecurityConfiguration.AutoAcceptUntrustedCertificates)
     {
       config.CertificateValidator.CertificateValidation += new
CertificateValidationEventHandler(CertificateValidator_CertificateValidation);
     }

// spustenie serveru
server = new SampleServer();
await application.Start(server);

// spustenie vlákna na sledovanie serverového stavu
status = Task.Run(new Action(StatusThread));

// výpis upozornení z udalostí vytvorených v relácii
server.CurrentInstance.SessionManager.SessionActivated += EventStatus;
server.CurrentInstance.SessionManager.SessionClosing += EventStatus;
server.CurrentInstance.SessionManager.SessionCreated += EventStatus;
```

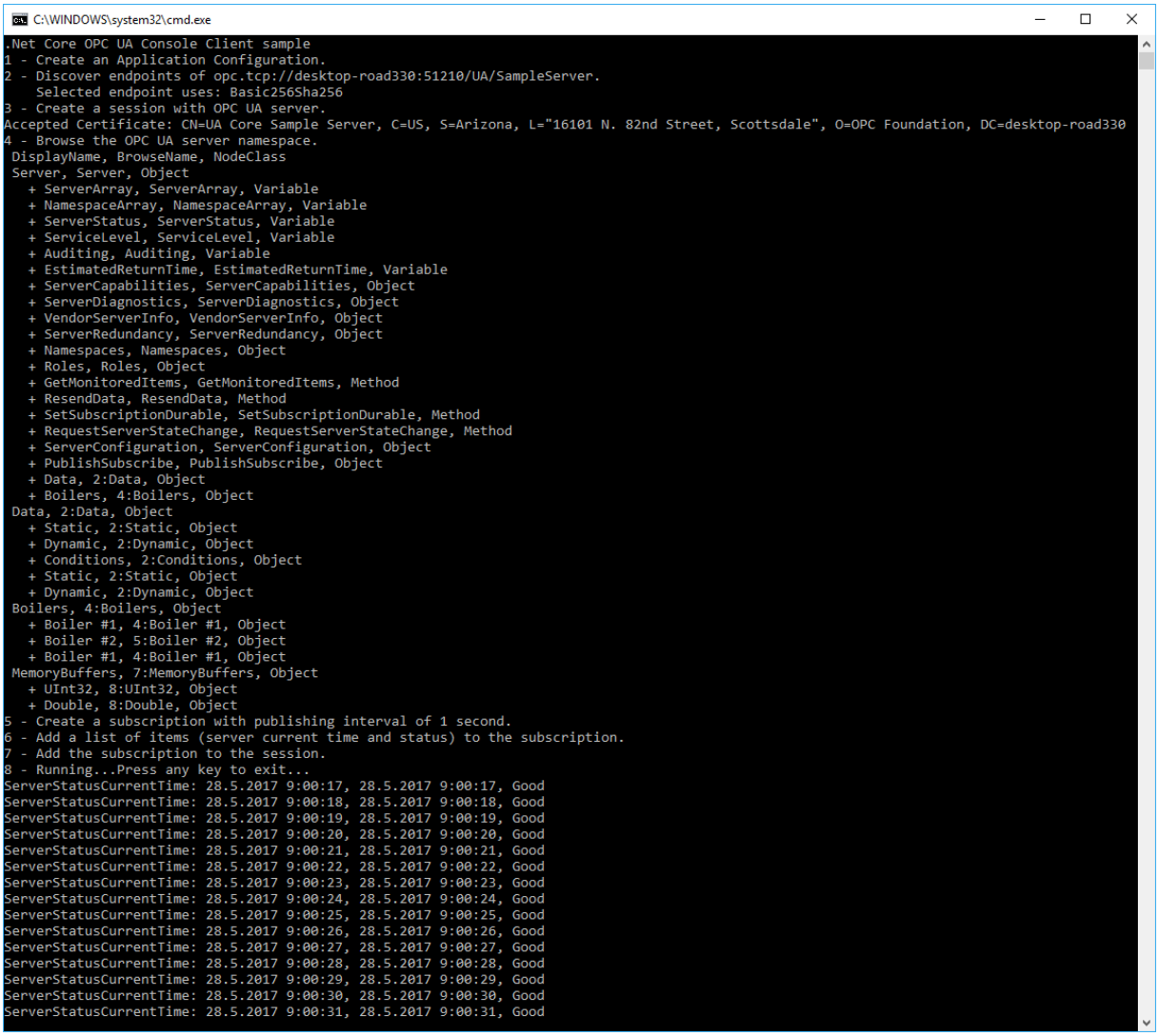
3.1.2 Validácia certifikátu

Nasledovný kód predstavuje funkciu na validovanie certifikátu. Na základe tejto validácie je potom umožnené alebo zakázané pripojenie klienta na server.

```
private static void CertificateValidator_CertificateValidation(CertificateValidator
validator, CertificateValidationEventArgs e)
        {
            if (e.Error.StatusCode == StatusCodes.BadCertificateUntrusted)
            {
                e.Accept = false;
                Console.WriteLine("Rejected Certificate: {0}", e.Certificate.Subject);
            }
        }
```

3.2 Client

Klientská aplikácia dokáže vyhľadať spustené servery na localhoste a pokúša sa pripojiť. Ak server schváli certifikát, aplikácia sa pripojí – vytvorí si reláciu na serveri. Následne si prezrie adresný priestor servera a potom vytvorí subscription, ktorá monitoruje uzol s id „i=2258“.



Obrázok 7: Ukážka spustenej klientskej aplikácie v .NET Core

3.2.1 Vytvorenie relácie

```
var session = await Session.Create(config, endpoint, true, ".Net Core OPC UA Console
Client", 60000, new UserIdentity(new AnonymousIdentityToken()), null);
```

3.2.2 Prehľadanie adresného priestoru

```
references = session.FetchReferences(ObjectIds.ObjectsFolder);

session.Browse(
  null,
  null,
  ObjectIds.ObjectsFolder,
  0u,
  BrowseDirection.Forward,
  ReferenceTypeIds.HierarchicalReferences,
  true,
  (uint)NodeClass.Variable | (uint)NodeClass.Object | (uint)NodeClass.Method,
  out continuationPoint,
  out references);

foreach (var rd in references)
{
  Console.WriteLine(" {0}, {1}, {2}", rd.DisplayName, rd.BrowseName, rd.NodeClass);
  ReferenceDescriptionCollection nextRefs;
  byte[] nextCp;
  session.Browse(
  null,
  null,
  ExpandedNodeId.ToNodeId(rd.NodeId, session.NamespaceUris),
  0u,
  BrowseDirection.Forward,
  ReferenceTypeIds.HierarchicalReferences,
  true,
  (uint)NodeClass.Variable | (uint)NodeClass.Object | (uint)NodeClass.Method,
  out nextCp,
  out nextRefs);

  foreach (var nextRd in nextRefs)
  {
    Console.WriteLine("   + {0}, {1}, {2}",
       nextRd.DisplayName,
       nextRd.BrowseName,
       nextRd.NodeClass);
  }
}
```

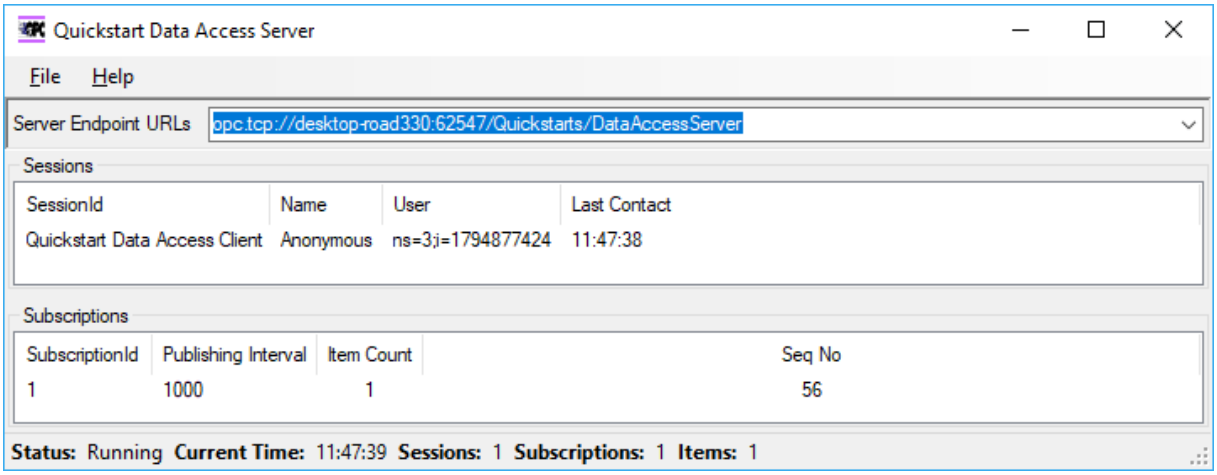
3.2.3 Vytvorenie subscription

```
var subscription = new Subscription(session.DefaultSubscription) {
  PublishingInterval = 1000
};

Console.WriteLine("6 - Add a list of items (server current time and status) to the
subscription.");

var list = new List<MonitoredItem> {
  new MonitoredItem(subscription.DefaultItem)
  {
    DisplayName = "ServerStatusCurrentTime",
    StartNodeId = "i=2258"
  }
};

list.ForEach(i => i.Notification += OnNotification);
subscription.AddItems(list);

Console.WriteLine("7 - Add the subscription to the session.");
session.AddSubscription(subscription);
subscription.Create();
```

4 .NET 4.5 aplikácia

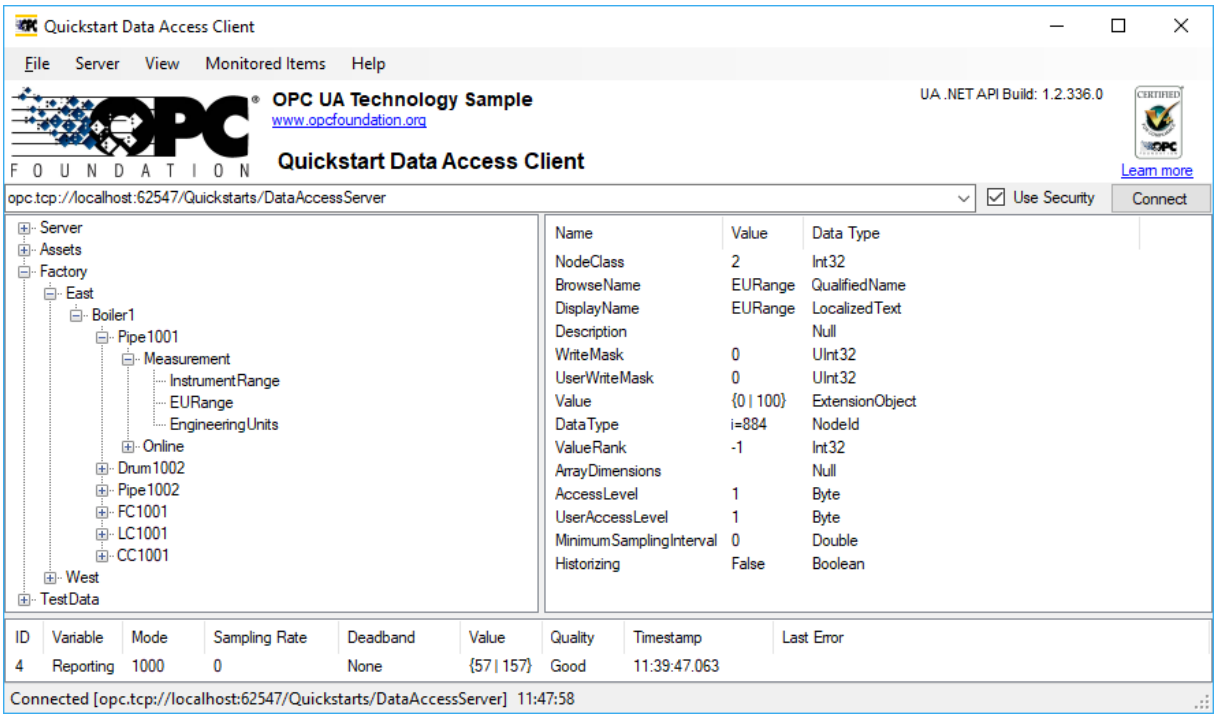
V tejto časti si predstavíme komplexnejšiu aplikáciu s grafickým rozhraním, ktorá je vytvorená výlučne pre operačný systém Windows. Na spustenie sme použili vývojárske prostredie Visual Studio 2017. Nakoľko je aplikácia rozsiahla, programový kód nebudeme popisovať a odkaz na celý projekt nájdete v referenciách.

Nasledovný obrázok zobrazuje serverovú aplikáciu, kde vidíme adresu servera, relácie a subscriptions. Na serveri beží relácia jedného klienta, ktorý vytvoril jednu subscription, ktorou žiada, aby server každú sekundu posielal nejaké informácie.



Obrázok 8: Náhľad na serverovú aplikáciu v .NET 4.5

Poďme sa pozrieť na klientskú aplikáciu, kde môžeme vidieť adresu serveru na ktorý sme pripojení a celý adresný priestor serveru. V klientskej aplikácii sme vytvorili monitorovanie uzlu „Engineering Units“ čiže subscription na serveri zasiela informácie o tomto node. V spodnej časti aplikácie môžeme vidieť monitorované uzly.



Obrázok 9: Náhľad na klientskú aplikáciu v .NET 4.5

5 Záver

Ukázali sme si, ako je možné vytvoriť jednoduchú aplikáciu v .NET Core a aj náhľad, na riešenia, aké je možné pomocou existujúcich technológií vytvoriť. Okrem uvedených aplikácii sme skúšali spustiť aj aplikáciu pre Universal Windows Platform, avšak projekt je spustiteľný len v rozhraní Visual Studio 2015 a v novšej verzii (2017) projekt nebolo možné spustiť kvôli nekompatibilite.

Zdroje

https://github.com/OPCFoundation/UA-.NETStandardLibrary

https://github.com/OPCFoundation/UA-.NET

https://opcfoundation.org/developer-tools/specifications-unified-architecture