Földvári András:

Megbízható kiber-fizikai rendszerek tervezése és ellenőrzése

Konzulens:
Dr. Pataricza András egyetemi tanár,
BME Méréstechnikai és Információs Rendszerek Tanszék

TDK dolgozat
2017

Tartalomjegyzék

Ábrajegyzék és táblázatjegyzék

Kivonat

A kiber-fizikai rendszerek (Cyber-Physical System – CPS) egyre fontosabbak és pótolhatatlanok az élet több egymástól lényegesen különböző területein, legyen az akár az egészségügy, szállítmányozás, okos gyártás vagy a biztonságtechnika. Az új funkcionalitásokra való igény lehetővé teszi a technológia fejlődését az érintett iparágakban, megfelelő tervezéssel és kivitelezéssel biztosítva a megbízhatóságot és a biztonságot. A kutatásom célja egy módszertan létrehozása volt, amivel követelmény vezérelt módon létrehozható egy megbízható, működő és biztonságos CPS. A módszertan algoritmikus módszerrel támogatja a megfelelő technológia kiválasztását, hogy a rendszer követelményei teljesüljenek.

A CPS fizikai világ és az informatika összekapcsolódásaként jön létre. A fizikai világból származó adatok feldolgozásához szolgáló logika megosztható a lokális erőforrások és a kibertérbeliek között (felhő). A kibertér ugyanakkor megjelenhet adatforrásként is (pl.: publikus érzékelési adatok, mint a meteorológiai előrejelzések), illetve intelligenciaforrásként is (pl.: távoli adatfeldolgozási eljárások). Az adatok nem csak a rendszer belső működését befolyásolhatják, hanem a CPS be is avatkozhat a fizikai világba. A két világ közti kapcsolatot gyakran beágyazott rendszerek adják a kibertérbeli megoldások interfészeként. A CPS világában a szolgáltatások bővülése és kifinomultsága tovább növelhető az egyes CPS-k egy nagyobb rendszerré való egybefűzésével (System of Systems – SoS).

Miután a számítógépes vezérlés hibáit a fizikai világgal való kölcsönhatás akár katasztrofálissá is erősítheti, növekszik a megbízhatóság, interoperabilitás és biztonság iránti igény. A CPS megalkotása a tervezéstől fogva összetett feladat. A kritikus CPS rendszerekben a szolgáltatásminőség biztosításához szükséges a széleskörű funkcionális és extra-funkcionális követelmények kielégítése.

A mérnöki világban bevett gyakorlat a különböző módszertanok használata a rendszer tervezése során. Ezek lehetnek ajánlások vagy egy adott szervezet által bevált gyakorlatok, mint a NASA System Engineering Handbook vagy az NIST CPS Framework. Hasonlóan, de-facto és ipari szabványok meghatároznak, illetve ajánlanak referenciaarchitektúrákat és platformokat, biztosítva az átjárhatóságot és az interoperabilitást a rendszerek és komponenseik között.

Különböző technológiák (pl.: Data Distribution Service – DDS, Blockchain, MQTT) hasonló funkcionalitások mellett más-más extra-funkcionális tulajdonságokkal (pl.: biztonság, áteresztőképesség, időszerűség, letagadhatatlanság) rendelkeznek. A megfelelő technológia kiválasztása kulcs a megfelelő szolgáltatás garantálásához. A szolgáltatások szintjének biztosításához kulcsfontosságú a követelmények teljesítése. A követelmények architektúrális leképzésének validációja és verifikációka döntő tényező a rendszer tervezésének fázisában.

Célom a tervezési fázisban végrehajtandó ellenőrzési tevékenységek algoritmikus támogatása volt. A szóba jövő illetve kiválasztott architektúra leképezhető kényszerek hálózatára. A hálózatban az egyes csomópontok megmutatják, hogy azok mennyire felelnek meg a követelmények egyes aspektusainak (biztonság, időszerűség, megbízhatóság, biztonságosság, integritás és letagadhatatlanság).

Az így létrejövő kényszerkielégítési probléma (CSP – Constraint Satisfaction Problem) garantálhatja a megfelelőséget egy egyedi megoldás ellenőrzésével vagy akár megadhatja az összes szóba jövő megoldást is.

A fenti módszerek lehetővé teszik a garantált minőségű CPS tervezését algoritmikus úton. A módszer segíthet a költségek és a tervezési idő csökkentésében.

Abstract

Cyber-Physical Systems (CPS) become increasingly important and essential in a variety of domains, like healthcare, transportation, smart manufacturing, and security. New functionalities enable technological advances in critical areas demanding a correct design and operation in order to assure dependability and security. My research objective was creating a methodology for the requirement-driven design of reliable, active and secure CPS solutions by matching requirements and implementation candidates in an algorithmic way.

CPSs are smart systems integrating physical and computational components. The application logic for physical data is shared between the local resources and the remote components (Cloud). Remote components work even as public data sources (e.g. public weather data) or intelligence sources (e.g. remote data processing procedures). The connection between the physical world and cyber-space is often established by embedded systems. The extensity and sophistication of services in the CPS world can be further increased by interconnecting individual CPSs to a larger one (System of Systems - SoS). The global demand for the reliable, interoperable and secure systems is rising.

To build and manage CPS is very complex. A proper guarantee of the quality of service delivered by a critical CPS necessitates the fulfillment of a wide spectrum of functional and extra-functional requirements.

System engineering has a variety of well-established design methodologies corresponding to the best industrial practice like NASA System Engineering Handbook or the NIST CPS Framework. Similarly, de-facto and industrial standards defining different reference architectures and platforms exist ensuring composability and interoperability of component subsystems and systems.

Different technologies (e.g. Data Distribution Service – DDS, Blockchain, MQTT) provide similar functionalities at different level of extra-functional properties (e.g. security, throughput, timeliness, non-reputability). Selection of the proper technology is a key to guarantee a proper service level of the CPS. This way the compliance to requirements is the key factor of service assurance. Validation and verification of the appropriateness of the requirement-architectural mapping is a decisive factor in the design workflow.

My target was to support the design-space exploration process by an algorithm helping the designer to maintain a clean view on the alternate architectural candidates. The designated architecture is mapped to a network of constraints. Here each node describes a particular system-component from the point of view of its capabilities to comply with the individual requirement aspects (Security, Timeliness, Reliability, Safety, Integrity, and Non-reputability).

The resulting Constraint Satisfaction Problem (CSP) can solve the compliance problem by checking an individual solution or enlisting all the compatible solutions.

The support facilitates the composition of correct-by-design CPS in an algorithmic way. These steps could help to reduce the cost and the design time.

1. Bevezető

A dolgozat egy megbízható és biztonságos kiber-fizikai rendszer (*CPS = Cyber-Physical System*) szisztematikus megtervezését és ennek nyomán egy olyan megvalósítási technológia kialakítását tűzte ki célul, amely használatával megalkotható egy, az összetett CPS-ekkel szemben felmerülő követelményeknek maradéktalanul megfelelő rendszer. A megalkotott módszerek algoritmikus módon segítik a tervezési folyamatot, segítségükkel már a tervezési fázisban garantálható, hogy a megrendelők az elvárásaiknak megfelelő rendszert, kapják kézhez.

A rendszerben lévő komponensek összeilleszthetőségének (kompozicionalitás) ellenőrzése és az extra-funkcionális követelmények teljesítése kihívást jelent a tervezés és megvalósítás szempontjából, ezért szükség lehet egy olyan módszerre, amivel biztosítható ezek meglétének ellenőrzése.

A bevezető fejezet a CPS-ek áttekintését és követelményeik vizsgálatát tűzte ki célul. A követelményekben megjelennek a CPS tervezése során felmerülő különbségek más rendszerekhez képest. A CPS-hez kapcsolódó logikai, időbeli, elérhetőségi és interoperabilitási kérdések is helyet kapnak.

Végül az esettanulmányi példa és a dolgozat felépítése olvasható. A bemutatott esettanulmány alapul fog szolgálni a szemléltető példákhoz.

1.1. Kiber-fizikai rendszerek

1.1.1. Áttekintés

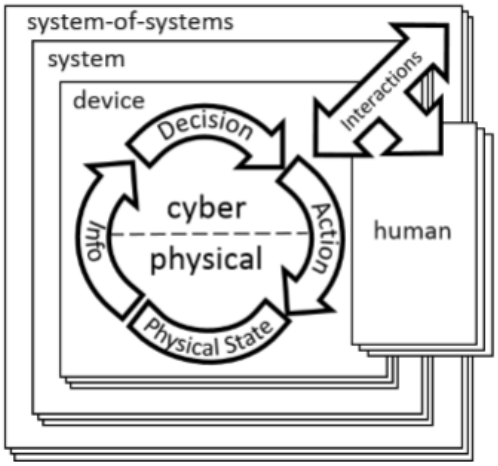
A CPS-ek egyre fontosabbak és pótolhatatlanabbak az élet több egymástól lényegesen különböző területén, legyen az akár az egészségügy, akár az autóipar. Miután a terület még csak kialakulóban van még a megnevezések és egyéb terminológia sem alakult ki. A dolgozatban ezért a [6] fogalomrendszerét használom tekintettel arra, hogy annak tárgya és szemlélete a dolgozat céljához közel esik és ez legalább önmagában konzisztens.

A CPS-ben felmerül a szenzorok hálózatba való bekötése, az adatok továbbítása és feldolgozása a vezérlés okán. Létrehozására és üzemeltetésére egyre nagyobb igény mutatkozik. A kritikus szolgáltatásokat nyújtó CPS szolgáltatásminőségének garantálásához a tervezésük során a funkcionális követelmények mellett több *extra-funkcionális (Extra-Functional Requirements = követelmények, amik meghatározzák a rendszer általános minőségét és tulajdonságait*) követelménynek is egyidejűleg eleget kell tenni.

A CPS-ek a fizikai világ és az informatika összekapcsolódásaként jönnek létre. A fizikai világból származó adatok feldolgozására szolgáló logika megosztható a lokális erőforrások és a kibertérbeliek között (*felhő (Cloud computing) = általában távoli szolgáltatás, amit az adatok tárolására, kezelésére, feldolgozására használnak, ezzel növelve a lokálisan elérhető korlátozott számítási és tárolási kapacitást*). A kibertér ugyanakkor megjelenhet adatok forrásaként is (pl.: publikus érzékelési adatok, mint az időjárásiak). A fizikai világ és a kibertér között kölcsönhatás lép fel, azaz az adatok befolyással lehetnek a rendszer működésére és fizikai világba való beavatkozás is lehetséges a kibertér által. A két világ közti kapcsolatot gyakran beágyazott rendszerek adják. A CPS-ek összetett rendszerek, állhatnak rendszerek rendszeréből (*SoS = System of Systems, Rendszerek egymásba ágyazása, összekapcsolása*) és az internethez is csatlakozhatnak (pl.: *Internet of Things = mindennapi eszközökbe beágyazott számítástechnikai eszközök összekapcsolása, amik lehetővé teszik adatok küldését, fogadását az interneten keresztül*).

A CPS-ek tartalmaznak *érzékelő* (*Sensor = eszköz, ami megfigyeli a fizikai világ valamelyik tulajdonságát és digitalis módon reprezentálja*), *beavatkozó* (*Actuator = eszköz, ami képes megváltoztatni egy elem fizikai*

tulajdonságát a bemenetének megfelelően), kommunikációs és logikai komponenseket is. Ezek biztosítják az időzítésérzékeny funkciókat és a meglehetősen sokféle beavatkozási lehetőséggel.



Ábra 1-1 - CPS konceptuális felépítése (átvéve: [10]–ből)

A CPS konceptuális felépítésén látszik, hogy a rendszerek között és a rendszereken belül is folyamatos interakció folyik a komponensek és a felhasználók között. Vannak olyan területek, ahol elengedhetetlen a rendszerrel való folyamatos kapcsolat, akár csak a megfigyelés, akár beavatkozás szempontjából.

A beavatkozás (*Action*) során létrejött fizikai (*Physical State*) változás érzékelését (*Information*) lehetővé téve megoldható, hogy az így kapott információ alapján (*Decision*) történjenek meg a további beavatkozások.

Mint minden rendszerben, a CPS-ben is fel kell tárni a rendszerre vonatkozó követelményeket, amik alapján megkezdhető az architektúrális tervezés folyamata. A következő részben a CPS-ben előforduló követelményekről és azok feltárását és rendszerezését segítő eszközökről lesz szó.

1.1.2. Követelmények

A hagyományos tervezési módszerek nem mindig felelnek meg a CPS tervezése során. Az összetett és kiterjedt funkcionalitást biztosító CPS nem minden aspektusában feleltethető meg más igényeket célzó rendszereknek. Előfordulhatnak követelmények, amik kifejezetten a CPS-re jellemzőek. Ez fakadhat a fizikai világgal való kapcsolatból vagy a több rendszer összekötésére való igényből. Kompatibilitási, időzítési és kommunikációs problémák is gyakran előfordulhatnak egy nagy rendszer esetén.

A tervezést úgy kell végrehajtani, hogy a későbbiek során lehetőség nyíljon új rendszerek csatlakoztatására és részrendszerek hozzáadására. Előfordulhat az is, hogy egy komplett rendszernek boldogulnia kell a több különböző alkalmazási tartományból érkező adatok továbbításával és feldolgozásával.

A következő követelményekre és igényekre kell felkészülni:

- A fizikai világ és a számítástechnika kapcsolata a CPS lényege. A CPS széleskörű funkcionalitása a fizikai érzékeléstől kezdve, az adatok feldolgozásán és tárolásán át, a fizikai közegbe való beavatkozásig terjed, így mindegyik területen szükséges a megfelelő működés biztosítása.

- A CPS gyakran rendszerek rendszerében is használatos. A különböző rendszerek között lehetnek időzítési és erőforrásbéli különbségek is. Ezek kezelésére is fel kell tudni készíteni az egész rendszert.
- Szükség van módszerekre, amikkel biztosítható az interoperabilitás. Interoperábilitás alatt értendő az, hogy a rendszerben és a rendszerek között közlekedő adatokat a komponensek hogyan értelmezik, és az, hogy a részrendszerek közti kommunikáció biztosított-e.
- A rendszer bővíthetőségét is biztosítani kell. Egy nagy rendszer részrendszereinek átjárhatósága, új részrendszerek hozzáadása és az ezek közti megfelelő működés biztosítása alapvető kell, hogy legyen.
- Kritikus rendszerek meghibásodása vagy a szolgáltatás kimaradása esetén komoly üzleti károk keletkezhetnek vagy emberéletek foroghatnak kockán. A szolgáltatások elérhetőségét a megbízhatóság (az első hibáig eltelő idő), illetve a rendelkezésreállás (annak valószínűsége, hogy a szolgáltatás elérhető és helyesen működik) jellemzik. A megbízhatóságot lehet biztosítani megfelelő tervezéssel, implementálással és karbantartással, de a rendelkezésreállás akár redundanciával is biztosítható.
- Számba kell venni a már használatban lévő eszközök új módon történő felhasználását, az adatok több nézőpont szerinti elemzését (pl.: az energiafelhasználási adatok jelezhetnek rendellenes működést is).
- Lehetővé kell tenni több alkalmazásterület összekapcsolását (pl.: az egészségügy és az okos közlekedés összekapcsolása segítheti a mentőautók mozgását és időben való megérkezését).
- A fizikai világba való beavatkozás magával vonja a megbízhatóság kérdését. Szükséges a
    - **biztonságot** (*= nem szándékos vagy illetéktelen hozzáféréstől való védelem az adtok megváltoztatása vagy megsemmisítése elleni védelem*),
    - **biztonságosságot** (*= a rendszer működése anélkül, hogy közvetve vagy közvetlenül sérülne a vagyon vagy a környezet, esetleg fizikai sérülés vagy egészségkárosodás lépne fel*),
    - **titkosítást** (*= adatok titkosítása*),
    - **integritást** (= *a rendszer helyes adatot szolgáltat, hiba és/vagy rosszindulatú támadás esetén is*),
    - **megbízhatóságot** (*= a rendszer vagy komponensének elvárt működése a megfelelő körülmények között egy meghatározott időben*) és a
    - **rugalmasságot** (= rendszer megfelelő skálázhatósága és bővíthetősége)

  nyújtani az összekapcsolt rendszerekben.
- A CPS-nek szabadon alakíthatónak kell lennie. Fel kell készülni a rendszer dinamikus változásaira, amik akár futásidőben is előfordulhatnak (pl.: új komponensek kerülhetnek a rendszerbe vagy IP címek változhatnak meg).
- Képesnek kell lennie többféle eszközzel és rendszerrel való kompatibilitásra, a kis teljesítményű érzékelőktől kezdve a nagy számítási teljesítménnyel rendelkező számítógépekig bezárólag.
- Több kommunikációs módot kell támogatnia (pl.: egy nagy rendszerben előfordul, hogy a komponensek közti kommunikációra más protokoll használatos, mint a rendszerek közti üzenetváltásokra).
- Az *időbeliség* (*Timeliness = a rendszer minden esetben egy felhasználói igényre vagy bementi változásra az előírt határon belül reagál*) egy központi tervezési kérdés. A fizikai világgal való kontaktus miatt gyakran előfordulnak olyan komponensek a rendszerben, amik viselkedésének időbeli helyessége kulcskérdés. Az időbeliség jellemzése a helyes reakcióidő, azaz, hogy a rendszer

minden esetben egy felhasználói igényre vagy bemeneti változásra az előírt határon belül reagál, illetve az átbocsátóképesség, azaz, hogy a rendszer adott időegység alatt hány kérést tud kiszolgálni.

- A rendszerrel való kapcsolatnak többféle emberi aspektusa van. Az felhasználó vezérelheti a CPS-t, segítheti annak vezérlését, felhasználhatja a CPS-t és a belőle jövő adatokat, de a felhasználó akár a mérés és beavatkozás tárgya is lehet (pl.: orvosi eszközök).

1.2. Esettanulmány

A következő esettanulmány alapjául fog szolgálni a példáknak és a módszer működésének bemutatása is e szerint lesz ábrázolva. A feladat leírásában a kritikus követelmények kiemelésre kerülnek.

Megfelelő hőmérséklet biztosítása egy kémiai folyamat során

A feladat egy tartályban lévő folyadék *hőmérsékletének szabályozása*. A folyadék hőmérsékletét egy (vagy több) tartályban elhelyezett *szenzorral* szeretnénk mérni, amikkel közvetlen megfigyelés tehető. A folyadék fűtéséért egy elektronikusan *vezérelhető fűtőtest* felelős. Az intelligens fűtőtest a szenzor által mért adatok alapján tudja a fűtési *hőmérsékletét beállítani*.

A folyadék molekuláris szerkezete túlmelegedésének hatására roncsolódik, így jelentős *anyagi kárt* okoz a cégnek. Nem megengedhető, hogy a folyadék hőmérséklete elérjen egy bizonyos *küszöbszintet*, ezért erre figyelemmel kell lenni. *Túlmelegedés* esetén *figyelmeztetni* kell a munkásokat és az operátort. A *figyelmeztetés* történhet *automatikusan* vagy az *operátor* közbenjárásával is.

A folyadék hőmérsékletét egy *operátor monitorozhatja*, ha szükségét látja, manuálisan be is avatkozhat a hőmérséklet állításába.

A hőmérsékletértékek analízise szempontjából szükséges a mért adatok eljuttatása a cég *felhőinfrastruktúrájába* úgy, hogy azok *letagadhatatlanok legyenek*.

Egy valódi rendszer során ennél lényegesen több igény merülhet fel, amiken kívül be kell tartani a törvényi előírásokat és szabályozásokat is. A példa célja egy kis rendszeren bemutatni a későbbiekben ismertetett módszer működését, amihez nem volt szükség egy valós, mindenre kiterjedő modell létrehozására.

1.3. A dolgozat felépítése

A dolgozat további három részből áll, amik a CPS tervezését és szolgáltatásbiztonságának ellenőrzését mutatják be.

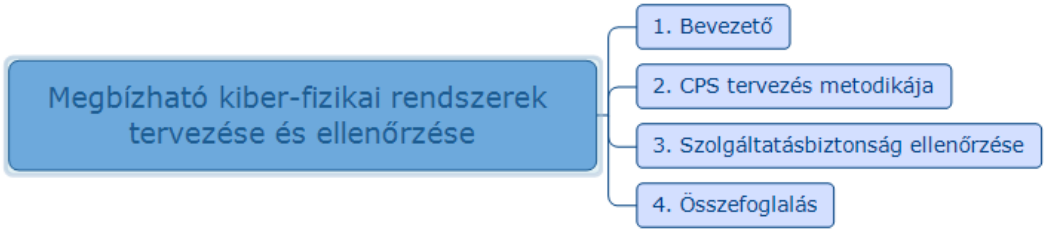
Szolgáltatásbiztonság alatt értjük azt a tervezési és működési garanciarendszert, amelynek alapján a felhasználó megalapozottan bízhat meg a rendszer nyújtotta szolgáltatások elérhetőségében és helyességében.

A második fejezet a CPS tervezését segítő keretrendszerek, referenciaarchitekrúrák bemutatásával kezdődik és a CPS implementálása során gyakran használt technológiák ismertetésével, összehasonlításával és a CPS-ben való felhasználásukkal ér véget.

Az harmadik fejezet foglalkozik a megtervezett CPS minőségének ellenőrzésével. Ennek keretében a módszer matematikai háttere és az algoritmikus folyamat kerül bemutatásra. Ezeken kívül ismertetésre kerül, hogy hogyan lehet a módszert integrálni egy széles körben használatos rendszertervezői

modellezőhöz, a SysML-hez. A módszer automatizálásához használható eszközök is itt kerülnek bemutatásra.

Végül a továbbfejlesztési és bővíthetőségi lehetőségről lesz szó, amik javíthatják a módszer hatékonyságát, használhatóságát és újabb funkciókat is hozzáadhatnak.



Ábra 1-2 - Dolgozat felépítése

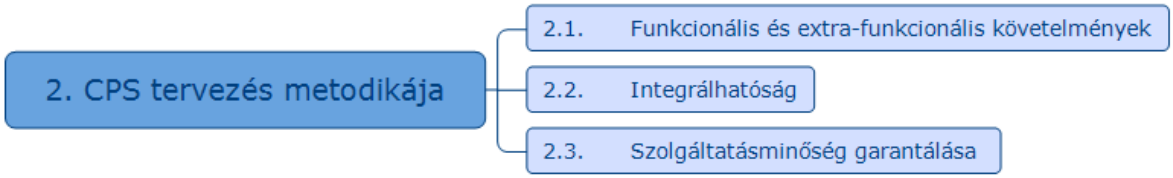
2. CPS tervezés metodikája

A fejezetben ismertetésre kerülnek azok az eszközök és módszerek, amik segítséget nyújthatnak egy jól specifikált CPS teljeskörű megtervezéséhez. A fejezet külön kitér a követelmények meghatározására, a rendszer architektúrális felépítésére és az implementáció során használható technológiákra.

Először a funkcionális és extra-funkcionális követelményekről lesz szó. Ezek megfelelő meghatározása elengedhetetlenül szükségesek a rendszer tervezésének további fázisaiban.

A rendszer integrálhatóságának és interoperabilitásának biztosításának feltétele a megfelelő architektúra kialakítása. Az architektúra meghatározását nagyban megkönnyíthetik általános érvényű CPS referenciaarchitektúrák, amik használatával mind a megrendelő mind pedig a fejlesztő számára megkönnyíthető a kommunikáció a rendszer felépítését és funkcionalitását illetően.

A fejezetben végül, azon technológiák bemutatására kerül sor, amik biztosítani tudják azokat a megfelelő szolgáltatásokat és funkciókat, amik elvárhatók egy CPS-től.



Ábra 2-1 - CPS tervezés metodikája - felépítés

2.1. Funkcionális és extra-funkcionális követelmények

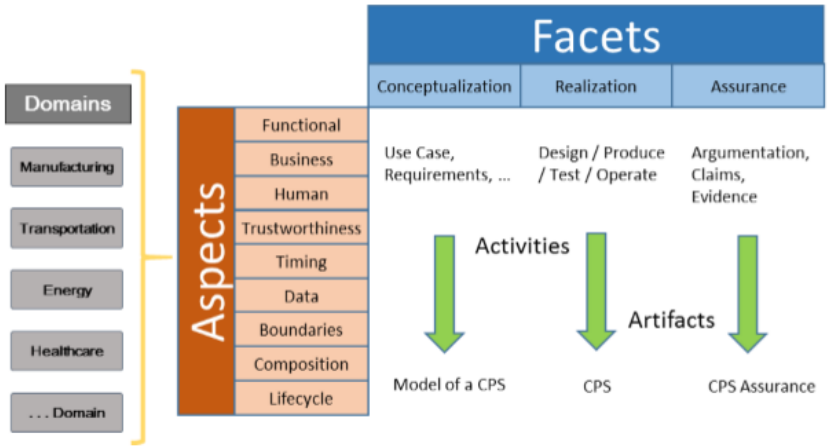
Egy rendszer tervezésének egyik legkorábbi és legfontosabb feladata a funkcionális és extra-funkcionális követelmények meghatározása. A korrekt és teljes követelményrendszer kialakítása kulcsfontosságú a rendszer működésének szempontjából. A követelmények alapján lehet finomítani a rendszert és meghatározni azokat komponenseket, amik a követelmények teljesítéséért felelősek lesznek.

Több sablon és módszer használható a követelmények meghatározásának segítésére. A dolgozatban egy kifejezettem CPS-re kialakított keretrendszer szolgált azok meghatározásának alapjául. A keretrendszer aspektusorientált módon kényszeríti ki a követelmények besorolását.

A következő szakaszban az NIST CPS Framework [10] kerül részletesebb bemutatásra, ami alapjául szolgálhat egy CPS követelményeinek szisztematikus meghatározásához és rendszerezéséhez.

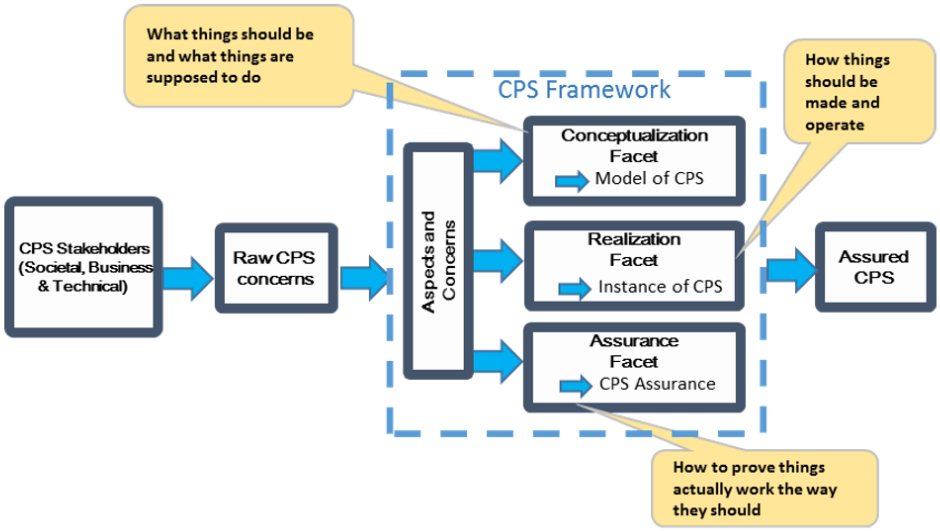
2.1.1. NIST CPS Framework

Az NIST (National Institute of Standards and Technology) CPS munkacsoport által kidolgozott keretrendszer célja a jellemzően bonyolult és sokaspektusú feladatból a megvalósítás alapjául szolgáló követelményrendszer strukturált kidolgozásának támogatása.



Ábra 2-2 - CPS Keretrendszer (átvéve: [10]–ből)

A követelményrendszer kialakításához az alábbi ábrán látható lépéseken keresztül lehet eljutni.



Ábra 2-3 - CPS Keretrendszer használata (átvéve: [10]–ből))

A keretrendszer használatának lépései a következők:

1. A CPS *domén* (*Application domain = specifikusan egy adott területre jellemző funkciók gyűjteménye és jellemzője*) meghatározása. Meghatározza a területet, ahonnan az érdekelt feleknek igényeik lehetnek a rendszerre nézve. Felmerülhetnek több szakterületre vonatkozó igények is.
2. Az igények horizontális felbontása (pl.: szociális, üzleti, technikai igények).
3. A horizontálisan felbontott igények analizálásával az igények besorolhatók a keretrendszer által meghatározott aspektusok alá.
4. Az igények dekompozíciója különböző módszerek segítségével történhet, melyek három megközelítésbe sorolhatók:
    - **Fogalmi megközelítés (Conceptualization) :** Magas szinten meghatározza a funkcionális követelményeket és a CPS kialakítását, ezzel megadva a CPS és a benne lévő komponensek működésének elvárásait.
    - **Megvalósítási megközelítés (Realization):** Meghatározza a rendszer részletes terveit (működésre és megvalósításra vonatkozóan), amik elengedhetetlenek a CPS megvalósítása szempontjából.
    - **Szavatolási megközelítés (Assurance):** Meghatározza, hogy a rendszerben lévő komponensek viselkedését milyen eszközökkel lehet biztosítani.

Az aspektusok lehetővé teszik a rendszerhez tartozó igények magas szintű csoportosítását. A keretrendszerben a következő aspektusok fordulnak elő:

- Funkcionális (Functional)
- Üzleti (Business)
- Emberi (Human)
- Megbízhatóság (Trustworthiness)
- Időzítés (Timing)
- Adat (Data)
- Határok (Boundaries)
- Módosíthatóság (Composability)
- Életciklus (Lifecycle)

A megközelítések és az aspektusok használatával a keretrendszer analitikai módszertant határoz meg a CPS-ek számára. Az aspektus alapú megközelítés az egész tervezés életciklusán keresztül középpontban van, függetlenül a használt fejlesztői megközelítéstől (pl.: vízesés modell, agilis vagy iteratív módszerek).

Az NIST célja egy referencia CPS leíró nyelv kialakítása is volt, ami alapjául szolgálhat különböző eszközöknek, szabványoknak és alkalmazásoknak.

2.2. Integrálhatóság

Mint, ahogy a CPS követelményeinél is látható volt, a CPS-ben kifejezetten fontos a rendszerek integrálhatóságának biztosítása. Ezek nem megfelelő kialakítása hatással lehet a rendszer egészének megfelelő működésére.

A CPS megfelelő követelményeinek kialakításán túl biztosítani kell az integrálhatóságot is. A bővíthetőség, a más rendszerekhez való integrálhatóság és az átjárhatóság a rendszerek között kulcskérdés. Ennek megtervezésére és az e funkcionalitásokat is támogató architektúra létrehozására biztosítanak keretet a referenciaarchitektúrák. Az IoT-ban érdekelt cégek elkészítették a saját megvalósításukat. Az Intel[1], az IBM [2]és a Microsoft [3]is rendelkezik saját architektúrával.

2.2.1. Referenciaarchitektúrák

A referenciaarchitektúrák általános iránymutatást nyújthatnak az újra felhasználható általános célú és ipari felhasználású architektúrák létrehozására. Biztosítható velük az implementáció függetlensége és a gyártósemlegesség. A referenciaarchitektúrák javaslatokat tehetnek az egyes komponensek felhasználására vonatkozóan.

Iránymutatást nyújtanak a felhasználók (megrendelő, fejlesztő) számára is, hogy megértsék a felhő nyújtotta lehetőségeket és előnyöket. Megkönnyíti a megrendelő és felhőinfrastruktúra szolgáltatója közti kommunikációt. A referenciaarchitektúrák általános biztos pontot nyújtanak ebben a gyorsan változó technológiájú környezetben.

A referenciaarchitektúrák kialakítása és az egyes részeinek felhasználási módjai nem írnak elő kötelességeket. Az ezektől való eltérés megengedett, legtöbb esetben szükséges is. Ennek ellenére megfelelő alapként tudnak szolgálni a rendszer általános architektúrájának kialakításához.

A referenciaarchitektúrák általában egyfajta rétegződést is leírnak, amiben meghatározhatják, hogy az egyes szolgáltatásokat hol érdemes megvalósítani. A komponensek ez alapján helyezhetők el a felhőinfrastruktúra valamely részében. Ez a tagolódás általánosan 3 részre osztható [26]:

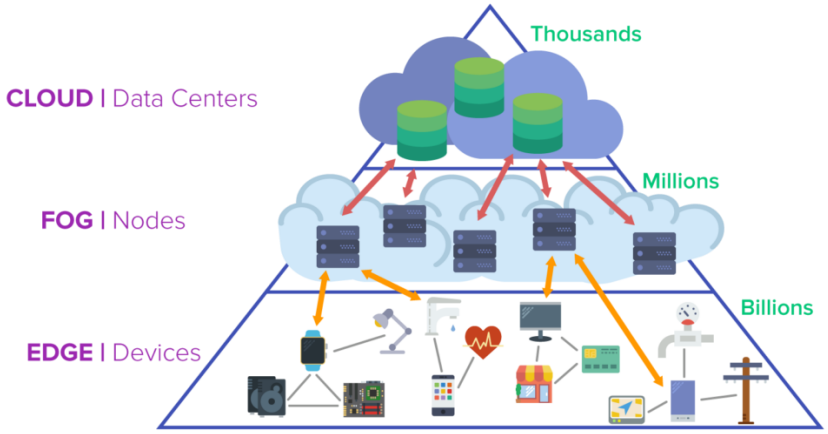
- **Edge**: Az architektúra legszélső rétege. Általában erőforráskorlátos eszközöket tartalmaz, amik lehetnek érzékelők (egészségügyi műszerek, fogyasztás mérők, mobil ezközökből származó adatok, autók által szolgáltatott adatok, stb.), beavatkozók (biztonsági berendezések, fizikai beavatkozók) és céleszközök (jelzőlámpa, kijelzők, szemétgyűjtő). Az eszközök között általában M2M (Machine to machine = gépek közti kommunikáció) kommunikáció alkalmazható. A real-time (valós idejű) eszközök szabályzását érdemes a rétegen belül megoldani elkerülve a magas si (= A késleltetés, ami alatt az adat a küldő csomópontból egy kommunikációs csatornán eljut a fogadó csomópontig.) időt.
- **Fog**: Az architektúra középső rétegében elkezdődhet az adatok feldolgozása. Nagyobb számítási teljesítményű eszközök találhatók itt. Ez a réteg még szoros kapcsolatban van az Edge réteggel. A kritikus időigényű számítások ebben a rétegben is elvégezhetők, itt még garantálható az alacsony késleltetés és a gyors feldolgozás. A szenzorokból származó adatok előfeldolgozása is megkezdhető a rétegbe elhelyezett logika segítségével. Az adatok szűrésével biztosítható a hálózat terheltségének csökkentése. Itt helyezhetők el útválasztók, amik a rendszer kapcsolatát biztosítják a publikus internethez.

[1] https://www.intel.com.au/content/www/au/en/internet-of-things/white-papers/iot-platform-reference-architecture-paper.html?cm_mc_uid=87665201247314890248265&cm_mc_sid_50200000=1502270832

[2] https://www.ibm.com/devops/method/content/architecture/iotArchitecture

[3] https://azure.microsoft.com/en-au/updates/microsoft-azure-iot-reference-architecture-available/?cm_mc_uid=87665201247314890248265&cm_mc_sid_50200000=1502270832

- **Cloud**: A architektúra tetején helyezkedik el a Cloud. Itt lehetőség nyílik az adatok további feldolgozására és tárolására. A központi tár biztosítja a megbízható és könnyű adatelérést az alsó (mind a Fog és Edge) rétegek számára. Kialakítható a felhasználói és üzleti alkalmazások logikája és az eszközök kezelésére és azonosítására szolgáló logika.



Ábra 2-4 - CPS architektúra rétegek (forrás: [24])

Az architektúra ezen kialakítása segít:

- Csökkenteni a hálózat terheltségét
- A mobilitás támogatásában
- Keretek közé foglalni az adatok alkalmazási és elhelyezkedési (architektúra réteg) szempontjait
- Elkerülni a SPOF (Single Point of Failure)-t, decentralizált és redundáns megoldások használatával

A referenciaarchitektúrák ettől a felépítéstől általában kisebb-nagyobb módon eltérnek. Új rétegeket hozhatnak be, bővíthetik a rétegek funkcionalitását vagy más megközelítések alapján bonthatják fel a rendszert.

A mintafeladat alapjául a CSCC (Cloud Standards Customer Council) referenciaarchitektúrája, a Cloud Customer Architecture for IoT szolgált. A következő részben ennek felépítése kerül bemutatásra.

2.2.1.1. CSCC - Cloud Customer Architecture for IoT

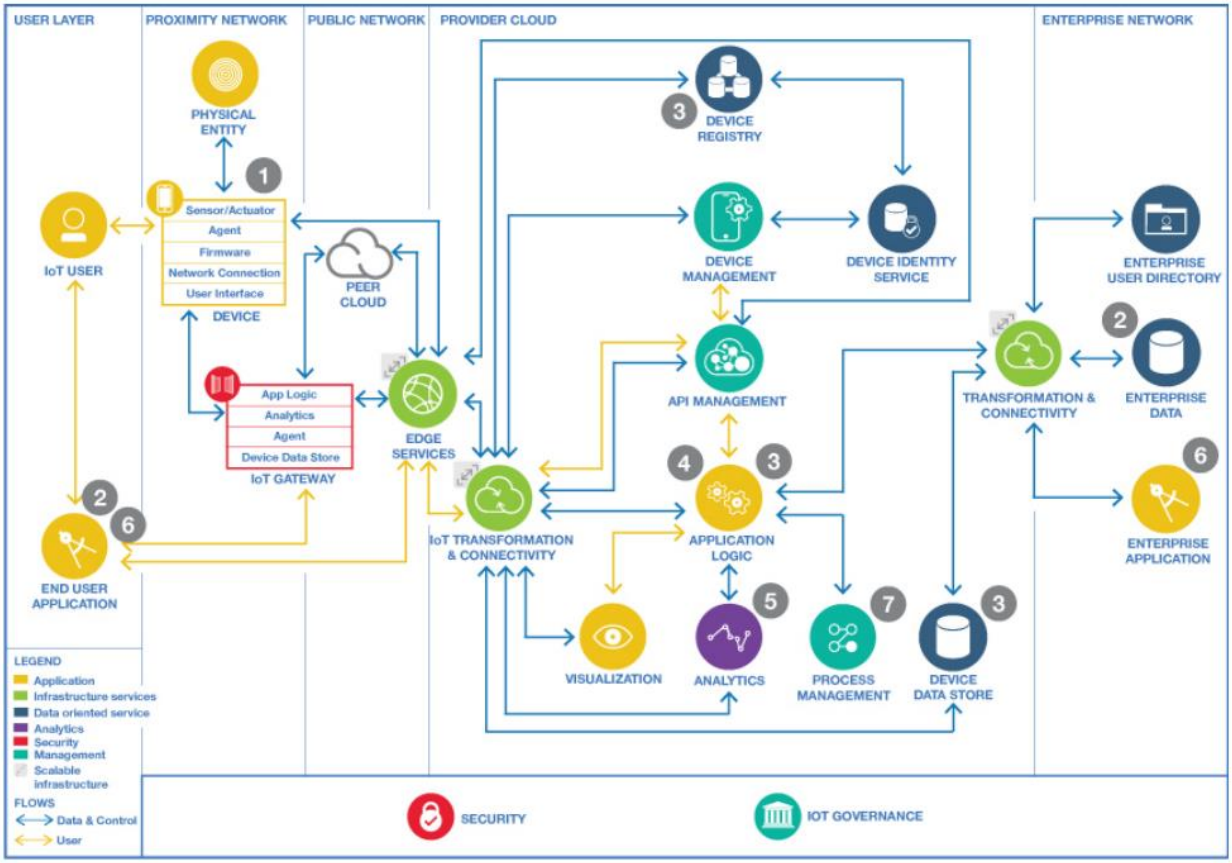
A CSCC (Cloud Standards Customer Council) végfelhasználói érdekképviseleti csoport [27], aminek célja a felhő alapú számítástechnika széles körű elfogadásának felgyorsítása, a felhőbe való átmenet szabványosításának segítése, valamint biztonsági és interoperabilitási problémákra vonatkozó szabványok lefektetése.

A megrendelő számára egy áttekinthető képet ad az architektúráról, ami biztosítja hogy a megfelelő szolgáltatást tudja igényelni.

A megrendelő és a fejlesztő számára egy közös alapot tud biztosítani, ami egy általános képet nyújt az architektúráról, ezzel megkönnyítve a felek közti kommunikációt és az igények és szolgáltatások jobb érthetőségét is biztosítja.

A referenciaarchitektúrával szemléltethető a szolgáltatások és komponensek elhelyezkedése a rendszer nézőpontjából és a speciális igényekre támaszkodva az egyes részrendszerek áthelyezése is megoldható.

(Elképzelhető, hogy a megrendelő számára nem megengedhető, hogy az ügyfél vagy alkalmazottainak adatai a publikus internetre kikerüljenek, így ezek elhelyezését a referenciaarchitektúrához viszonyítva módosítani kell.)



Ábra 2-5 - CSCC - Cloud Customer Architecture for IoT

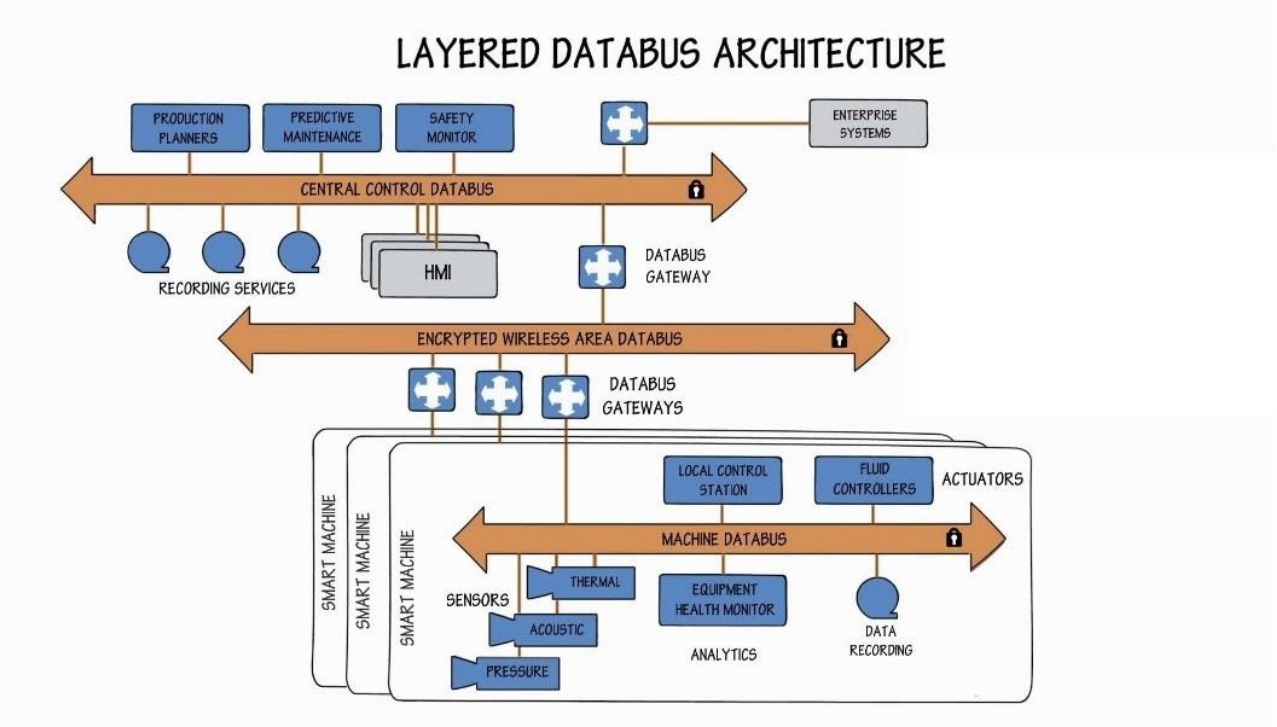
A Cloud Customer Architecture for IoT [28] referenciaarchitektúra irányt mutat az IoT támogatására a felhő alapú számítástechnika segítségével. A rendszer komponenseit három szintre sorolja (,ahogy az ábrán is látható):

- **Edge** (User Layer, Proximity Network): Ebben a rétegben helyezkednek el a szenzorok és a felhasználói vezérlő alkalmazások. Lehetőség nyílik az adatok helyben való feldolgozására, ezzel biztosítva a gyors válaszidőt. Lehetnek olyan adatok is a rendszerben, amiket nem szeretnénk a publikus interneten keresztül szállítani, ezek tárolására és feldolgozására is itt nyílik lehetőség.
- **Provider cloud** (Public Network, Provider Cloud): Ez szolgál a felhő alapú számítástechnika alapjául és hozzá kapcsoló szolgáltatásokat nyújt. Az itt elhelyezett szolgáltatások erősebb számítási teljesítmény tudnak biztosítani. Tartalmazhat eszközöket kezelő szolgáltatásokat és ezekre vonatkozó információkat. Az IoT rendszer kezelésére is szolgálhat és a feldolgozott adatok vizualizációjának elkészítésére is megfelelő, amit a végfelhasználókhoz eljuttatva kényelmesebb elemzési lehetőségeket biztosít.

- **Enterprise Network**: Az üzleti hálózatban találhatók meg az engedélyezett felhasználókra vonatkozó információk, mataadatok a rendszerre és a benne utazó adatokra vonatkozóan, a rendszerben összegyűjtött adatok nyers és feldolgozott változatai, tevékenységnaplók és az üzleti adatok megjelenítésére, kezelésére szolgáló alkalmazások.

2.2.1.2. Layered Databus Architechture

A világban egyre több területen az okosközlekedéstől az okosegészségügyön át az okosgyárakig mindent behálóznak az okos szenzorhálózatok, ezt a kialakult rendszert nevezzük *Industrial Internetne*k. [25] Az Industrial Internet biztonsági és időzítési követelményei jóval magasabbak, mint az általában. Ezen szenzorhálózatok integrációjához szükség van egy közös architektúrára. Az *Industrial Internet Consurtium (IIC)* több különböző iparágakban tevékenykedő nagy céggel összefogva egy általános referenciaarchitektúrát hoztak létre. Az Industrial Internet Refernce Architecture (*IIRA*) segít az interoperábilitás megvalósulását, technológiai segítséget nyújt és elősegíti a szabványok fejlődését. Az architektúra alapjául a Layered Databus Architecture (LDA) [9] (*Databus = adatközpontú információmegosztó technológia, ami virtuális globális adattéren keresztül valósítja a meg az alkalmazások közti adatcserét*) szolgál, ami manapság elterjedt az IoT rendszerekben. Ez biztosítja az alacsony késleltetést, biztonságos és közvetlen alkalmazások közötti kommunikációt. Az LDA-ban a résztvevők publish-subscribe minta alapján oszthatják meg az adatokat.



Ábra 2-6 - Layered Databus Architecture (átvéve: [9]-ból)

Az architektúrában a rétegek között átjárók (Gateway = Komponens aminek segítségével különböző hálózatokat lehet összekapcsolni.) biztosítják az üzenetek megfelelő konverzióját.

A legalsó architektúrális szinten helyezkednek el azok az alacsony intelligenciával rendelkező eszközök, amik lehetnek szenzorok vagy beavatkozók. Az időkritikus funkcionalitásokat igénylő komponenseket is itt érdemes elhelyezni (pl.: lokális vezérlés, egészségügyi monitorozó eszközök), azért mert itt nyílik lehetőség az adatok gyors mozgatására. Lokális adattárolásra is lehetőség nyílik, aminek segítségével az adatvédelem aspektusai valósíthatók meg.

A felsőbb szinteken nagyobb tárolókapacitású és feldolgozó képességű komponensek helyezkedhetnek el.

A rétegekben és a rétegek közti kommunikáció is lehet titkosított, akár vezeték, akár vezeték nélküli közeg felhasználásával.

Az adatbusz kulcsfontosságú jellemzői a következők:

- Az alkalmazások közvetlenül kapcsolódnak az adatokhoz
- Értelmezi és szűrheti az adatokat
- Az implementáció szabályokat ír elő és QoS (*Quality of Service = szolgáltatás minősége*) szolgáltatásokat biztosít, mint az adatátvitel, megbízhatóság és az adatok biztonsága

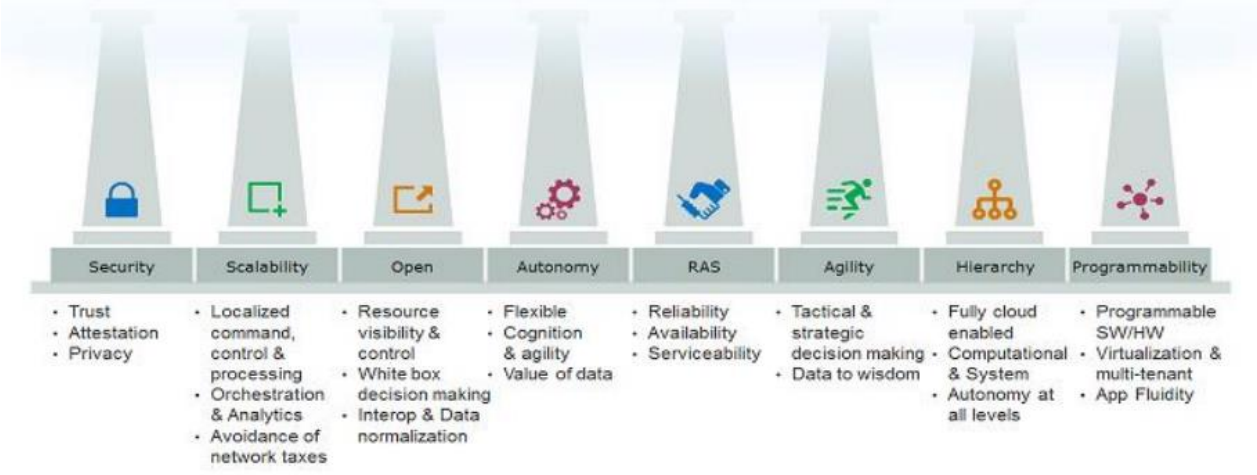
Az LDA használata a következő előnyökkel jár [9]:

- Gyors eszközök közti adatátvitel
- Automatikus adat és alkalmazás feltérképezés (akár adatbuszok között is)
- Robosztus működés – lehetővé teszi több ezer eszköz csatlakozását
- Magas rendelkezésreállás és ellenálló képesség
- Lehetővé teszi a komplex rendszerek tervezését

A LDA felépítését, előnyeit jellemzőit megvizsgálva látható, hogy alkalmas egy CPS architektúra kialakítására.

2.2.1.3. OpenFog

Az OpenFog [29] referenciarchitektúrát 8 elv vezérli. Ezek határozzák meg a rendszer legfontosabb jellemzőit. Az architektúrát egy horizontális megközelítés jellemzi.

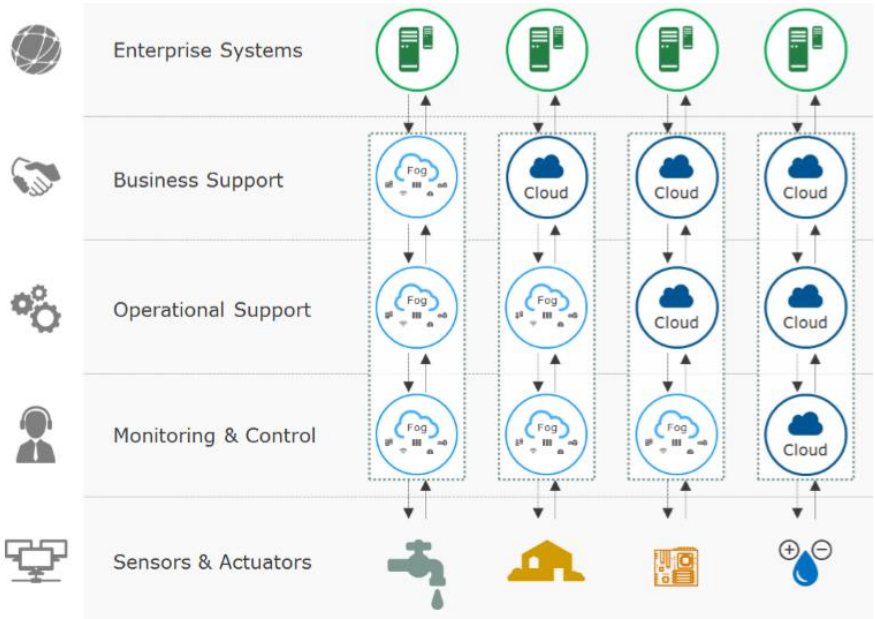


Ábra 2-7 - Az OpenFog referenciaarchitektúra elvei

Az architektúra alapjául szolágáló 8 elv a következő:

- **Security** (Biztonság)
- **Scalability** (Skálázhatóság)
- **Open** (Nyitottság)
- **Autonomy** (Autonómia)
- **RAS** (Reliability, Availability, Serviceability) (Megbízhatóság, Elérhetőség, Karbantarthatóság)
- **Agility** (Gyorsaság)
- **Hiearchy** (Hierarchia)
- **Programmability** (Programozhatóság, módosíthatóság)

Többféle architektúrális megközelítést nyújt, a rendelkezésre álló eszközök és az igényeknek megfelelően.



Ábra 2-8 -OpenFog referenciaarchitektúra használati modellek

Előfordulhatnak esetek, amikor a szolgáltatások Felhőbe való elhelyezése nem lehetséges (fizikailag nincs hozzáférés a publikus internethez, adatok gyors feldolgozására valóigény) vagy nem megengedett (szabályozások, adatbiztonság, katonai felhasználás). Ezek előfordulhatnak különböző szinteken:

- **Monitoring & Controll**: Időérzékeny adatok feldolgozásánál.
- **Operational Support**: Azoknál a folyamatoknál, ahol a működés központi szerepet kap.
- **Business Support**: Adatbiztonságra vonatkozó megkötések.

2.3. Szolgáltatásminőség garantálása

Ahhoz, hogy egy rendszerben a szolgáltatásminőségét garantálni lehessen, a megfelelő technológiák [15] felhasználására van szükség az implementálás során. Ezek kiválasztása már a tervezési fázisban meg kell, hogy történjen.

Számos a CPS-ben is használható megoldás létezik, amik biztosítani tudják a kifejezetten CPS specifikus követelmények megvalósítását.

A fejezetben a széles körben használt és a példafeladatban felhasznált technológiákról lesz szó. A példafeladatban használt megoldások részletesen is ismertetve lesznek.

2.3.1. Technológiák

A rendszerbe felhasznált technológiák választásához több szempontot is figyelembe kell venni. A technológiákat úgy érdemes kiválasztani, hogy azokkal teljesen lefedhetők legyenek a funkcionális és extra-funkcionális követelmények. A vizsgálat során figyelembe az alábbi szempontokat érdemes figyelembe venni:

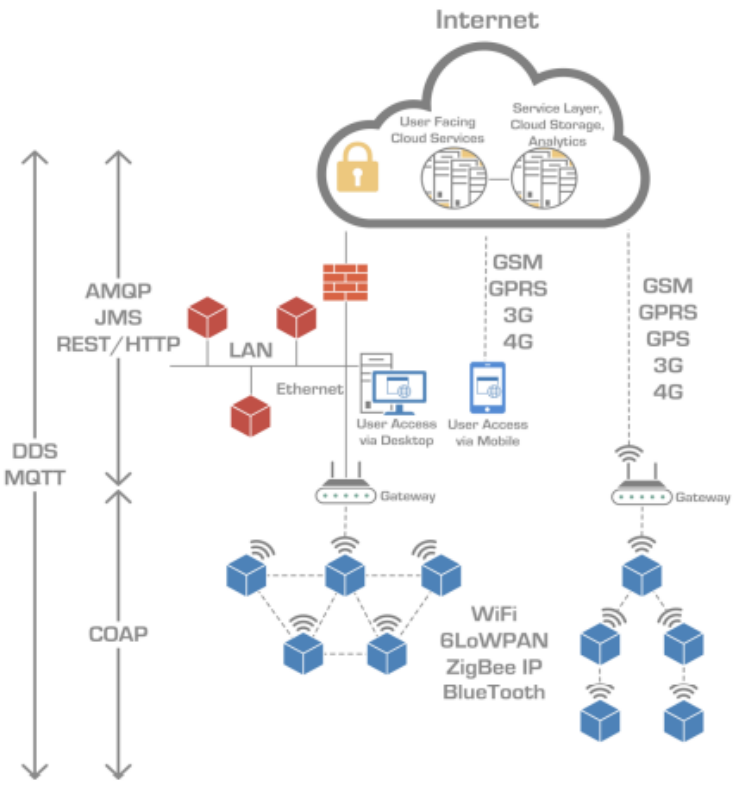
- **Tulajdonságok**: Milyen adatátvitelt, késleltetést, QoS szolgáltatásokat tud nyújtani?
- **Támogatott platformok**: Milyen platformokra érhető el az adott technológia? (pl.: beágyazott rendszerek, mobilok, kis számítási kapacitású rendszerek)
- **Biztonság**: Milyen titkosítási lehetőségek vannak?
- **Elterjedtség**: Milyen széles körben használják?
- **Elérhetőség (license)**: Mennyibe kerül az adott technológia használata?
- **Támogatás**: Egy adott megvalósításhoz a gyártó milyen támogatást nyújt? (gyors válaszidő, forum)
- **Alkalmazási terület**: Hol alkalmazzák a gyakorlatban?

Az alábbi táblázat egy általános áttekintést nyújt a CPS-ben is használható kommunikációs technológiákról.

	DDS	MQTT	AMQP	CoAP	REST	JMS
Architektúra	Globális adattér	Broker	Peer-to-peer Broker	Peer-to-peer	Peer-to-peer	Broker
Üzenetküldés	Publish-subscribe	Publish-subscribe	Publish-subscribe	Request-reply	Request-reply	Request-reply
Interoperabilitás	Igen	Részben	Igen	Igen	Igen	Nincs
Valós idejű működés	Igen	Nem	Nem	Nem	Nem	Nem
Szabvány	OMG RTPS és DDSI	OASIS Standard	OASIS AMQP	RFC	-	Java Community Process JMS
Mobil eszközök támogatása	Igen	Igen	Igen	HTTP-n keresztül	Igen	Java futtatására alkalmas eszközön
Dinamikus felfedezés	Van	Nincs	Nincs	Van	Nincs	Nincs
Biztonság	Vendor specfikus DDS-Security szabvány	TLS/SSL	SASL/TLS	DTLS	TLS/SSL	TLS/SSL
Alkalmazási területek	Valós idejű rendszerek	Szűkös erőforrású rendszerek	Üzleti alkalmazások	Vezeték nélküli kommunikáció	Web szolgáltatások	Back-end szolgáltatások
QoS	22	3	3	Visszajelzés kérhető az üzenetekről	TCP által meghatározott	3
Vendorok	Prismtech RTI OpenDDS	HiveMQ Mosquitto	Qpid SwiftMQ	Artik Cloud Autodesk Fusion Connect	-	ActiveMQ Amazon SQS

Táblázat 2-1 - Technológiák általános összehasonlítása

Az alábbi ábrán látható, hogy az egyes technológiák milyen architektúra rétegben használhatók a legcélszerűbben. Vannak olyanok, amik kifejezetten specifikusan két réteg között vagy rétegen belül nyújtanak megfelelő szolgáltatást (pl.: CoAP) és olyanok is, amik segítségével az egész CPS-en átívelő kommunikáció biztosítható (pl.: DDS, MQTT). Ezek alapos figyelembevételével érdemes kialakítani a megvalósítás során használni felhasználandó technológiákat.



Ábra 2-9 - Technológiák elhelyezése az architektúra rétegekben (átvéve [15]-ból)

AMQP – Advanced Message Queuing Protocol

Az AMQP egy üzenetközpontú protokoll, ami a pénzügyi szektorból emelkedett ki. Célja a nem interoperabilis és nem egységes üzenetküldési szolgáltatások leváltása volt. Biztosítja az átjárhatóságot a különböző vendorok implementációi között. Minden alkalmazás, ami képes az AMQP-nek megfelelő üzenetek kezelésére, képes lesz az együttműködésre a programozási nyelvtől függetlenül.

Az AMQP egy bináris (ember számára nem olvasható), alkalmazási rétegben elhelyezkedő protokoll. Több üzenetküldő alkalmazást és kommunikációs mintát támogat. Az üzenetközpontú kommunikációt kiegészíti QoS szolgáltatásokkal. Amik lehetnek:

- **At most once**,
- **At least once**, és
- **Exactly once** garancia az üzenetek kézbesítése során.

Hasonló QoS szolgáltatásokat biztosít a később bemutatott MQTT protokoll is, így ezek részletezése az ott olvasható.

JMS

A JMS egy széles körben elterjedt publish-subscribe alapú üzenetküldő protokoll. A Java Enterprise Edition (Java EE) részeként egy API-t (Application Programming Interface = Alkalmazásprogramozási felület)

biztosít az üzenetek küldésére a kliensek között. Lehetővé teszi az alkalmazáskomponenseknek, hogy üzeneteket küldjenek, fogadjanak és olvassanak. Elosztott alkalmazás esetén biztosítja a lazán csatolás elvének megvalósulását, a megbízhatóságot és az asszinkronitást. A publish-subscribe mintán kívül biztosítja a pont-pont összeköttetéseket is. Az AMQP-vel ellentétben a szabvány csak az API-t írja le, így az vendorok közti interoperabilitás nincs biztosítva.

REST

A REST lehetővé teszi a rendszerek közti interoperabilitást. Ennek a gyakran webszolgáltatásként használt technológiának a megvalósítása a kliens-szerver architektúrát követi, ahol a kliens kéréseket tud küldeni a szerver számára, akit kérhet egy művelet végrehajtására vagy adatok elküldésére.

Megfelelő metódusok meghatározásával lehetőséget nyújt a CPS-ben való használatra is. Leggyakrabban az Edge feletti architektúrarétegtől használják, az interneten való kommunikációra.

CoAP - Constrained Application Protocol

A CoAP egy szoftverprotokoll, ami erőforráskorlátos elektronikai eszközök számára lehetővé teszi a kommunikációt az interneten keresztül. A kommunikáció alapjául a REST modell szolgál, aminek köszönhetően fejlesztői szempontból a működése nagyban hasonlít a HTTP-hez. Egy szenzortól való adatlekérés nem sokban különbözik egy Web API használatától. Ez a megközelítés nagyban segíti az interoperablitás létrejöttét. Szállítási rétegben az UDP protokollt használja, elősegítve az alacsony energiájú és egyszerű eszközökön való használatát.

További technológiák

Az alábbiakban felsorolt technológiák szolgáltak a mintapélda megvalósításának alapjául. A választásnál szempont volt, hogy a választott technológiákkal az összes vizsgált követelményaspektus lefedhető legyen. Figyelembe kellett venni:

- IoT-ben előforduló széles skálán mozgó igények támogatását (eszköz „élénkségének” vizsgálata, megbízhatósági követelmények, szolgáltatáskiesés, adatok tárolásának támogatása, adatok szűrése)
- Az IoT eszközök szűkös számítási kapacitását és memóriáját
- Az adatok valódiságának, letagadhatatlanságának biztosítását (meghibásodáskor, jogi helyzetekben fontos lehet)

A választott technológiák lefedik a fent említett követelményeket és sok aspektusban átfedésben is vannak egymással, ezzel megadva a választási lehetőséget is.

2.3.1.1. DDS

Mi a DDS?

Az LDA egyik megvalósítása a *Data Distribution Service (DDS)* [5]. A DDS egy *middleware* (= *az operációs rendszer és az alkalmazás között helyezkedik el*) protokoll, ami publish-subscribe kommunikációt használva, különböző QoS szolgáltatásokkal támogatja a valós idejű, megbízható, robosztus alkalmazások

létrehozását. A DDS szabványt az *OMG* (Object Manage Group) [4]dolgozta ki. A piacon több implementációja is megtalálható, mint például

- az **RTI Connext DDS**[5],
- a **Prismtech Vortex DDS** [6]
- és az **OpenDDS**[7].

A szabvány részei

A szabvány alapvetően egy adatközpontú publish-subscribe modellt ír le, ami meghatározza az elosztott alkalmazások közti kommunikációt és integrációt. A szabvány meghatározza:

- az alkalmazásprogramozási felületet (API)
- és a kommunikációs szabályrendszert (viselkedés és szolgáltatásminőség (*QoS*)).

Ezeken kívül rendelkezik egy az implementációk interoperabilitását biztosító valós idejű összeköttető protokoll szabvánnyal (**DDSI-RTPS**), ami meghatározza az üzenetek szemantikáját és szintaktikáját.

A DDS szabvány kiegészítései és szolgáltatásai lehetőséget biztosítanak egy még bővebb funkcionalitással rendelkező rendszer létrehozására.

- **DDS-XTypes**: Bővíthető és dinamikus topikokkal való kiegészítés.
- **DDS-Security**: DDS biztonsági kiegészítése.
- **DDS-RPC**: Távoli eljáráshívásokkal való kiegészítés.
- **DDS-Web**: Platformfüggetlen módon biztosítja a web klienseknek a DDS rendszer elérését.

A DDS-t úgy tervezték, hogy bármilyen programozási nyelvvel használható legyen. A legtöbb implementációjában a következő nyelvek érhetők el:

- C/C++
- Java
- C#

Komponensek

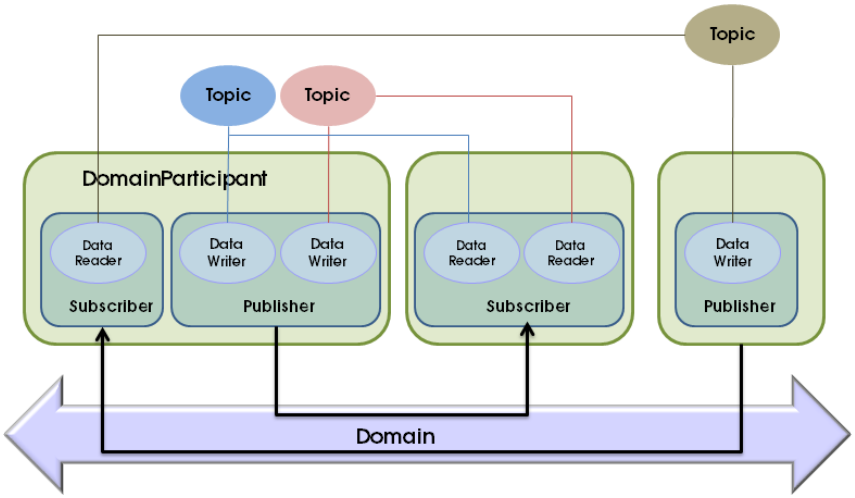
A DDS fő komponensei a következők:

- **Publisher**
- **DataWriter**
- **Subscriber**
- **DataReader**
- **Topic/FilteredTopic**

A Publisher kezeli a DataWritereket a Subscriber pedig a DataReadereket. A publisher és a subscriber oldal közti kommunikációs összeköttetést a Topic határozza meg. A Topic írja le az adat típusát is. A FilteredTopic

[4] http://portals.omg.org/dds/
[5] https://www.rti.com/
[6] http://www.prismtech.com/vortex
[7] http://opendds.org/

lehetőséget biztosít az adatok szűrésére, így gondoskodva róla, hogy mindenki a neki számára megfelelő adatot kapja meg.



Ábra 2-10 - DDS felépítése (átvéve: [30]-ból)

A fent említett komponensek egy **DomainParticipant**-hoz kapcsolódnak, a DomainParticipnat pedig egy **Domain**-hez kapcsolódik. Csak azok a komponensek képesek kommunikálni, amik ugyanazon Domain-hez csatlakoznak.

A QoS [14] határozza meg a szolgáltatás viselkedését és ez is befolyásolja a kommunikációs összeköttetést. A QoS hozzárendelhető a DDS komponenseihez, amiknél biztosítani szeretnénk az adott viselkedést. A teljes szabvány több mint 22 QoS szabályt ír le, mint például:

- **Deadline** – egy maximum időtartam, amin belül az adott elemnek frissülnie kell
- **Durability** – meghatározza, hogy a már publikált adatok eljussanak a később vagy újra-csatlakozó olvasókhoz
- **History** – meghatározza, hogy a rendszer hogyan és meddig visszamenőleg tároljon adatokat
- **Liveliness** – segítségével érzékelhető, ha egy komponens kiesik a rendszerből
- **Ownership** – a szabály segít a rendszerekben lévő redundanciát biztosító komponensek kezelésében
- **Reliability** – jelzi, ha egy adat elveszett a küldés közben

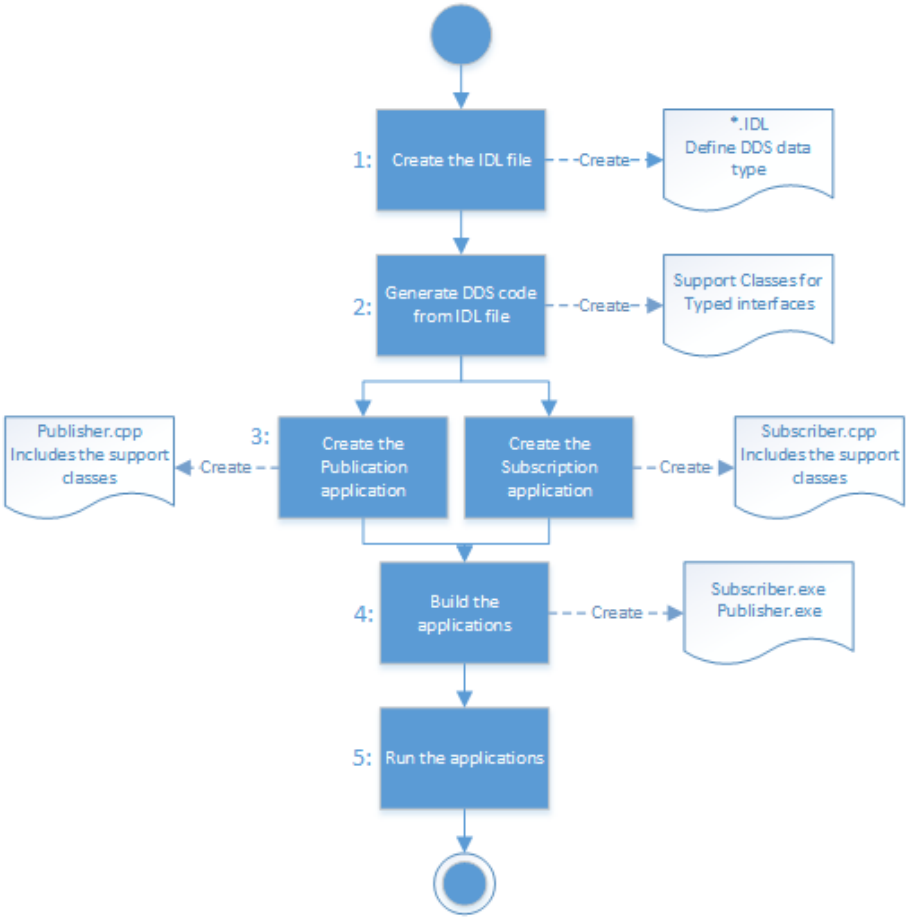
Ezeken kívül a DDS egyéb szolgáltatásokat is biztosíthat implementációtól függően. Szolgáltathatnak különböző hibakereső, monitorozó eszközöket vagy adatbázisintegrációt segítő kiegészítéseket.

DDS alkalmazásfejlesztés

A szabványban meghatározott API-val az alkalmazások könnyen elkészíthetők. Egy DDS alkalmazás implementálása során általánosan a következő lépéseket kell végrehajtani:

1. ***IDL (Interface Definition Language)* fájl létrehozása** – Meghatározza a DDS adat típusát
2. **DDS kód generálása az IDL fájlból** – Létrehozza a segédosztályokat

3. **A Publisher és a Subscriber alkalmazások elkészítése** – Szükséges a segédosztályok importálása
4. **Az alkalmazások fordítása** – Megvalósítás nyelvétől függő fordító használata
5. **Futtatás**



Ábra 2-11 - DDS alkalmazáskészítés lépései

DDS használata CPS-hez

A DDS fő komponenseit kiegészítve a szabvány által nyújtott QoS szabályokkal lehetőség van egy CPS alkalmazásban felmerült követelmények kielégítésére DDS segítségével.

A példa CPS-ben a következő komponensek fordulnak elő, szolgáltatásuk az egyes DDS komponensekkel biztosítható:

- **Szenzor** (Hőmérséklet mérése) – adatok küldése, Publisher segítségével
- **Beavatkozó** (Folyadék hőmérsékletének szabályozása) – adatok fogadása, Subscriber segítségével
- **Operátor dashboard** (Felügyelő vezérlőállomása) – adatok fogadása, vezérlőjelek küldése (Publisher, Subscriber, FilteredTopic)
- **Adatbázis** (Az adatok elemzéséhez) – Database Service segítségével biztosítható a működés

Látható, hogy a komponensek funkcióinak támogatása megoldható DDS segítségével. A QoS szolgáltatásokkal kiegészítve biztosíthatók az időbeli, megbízhatósági követelmények is.

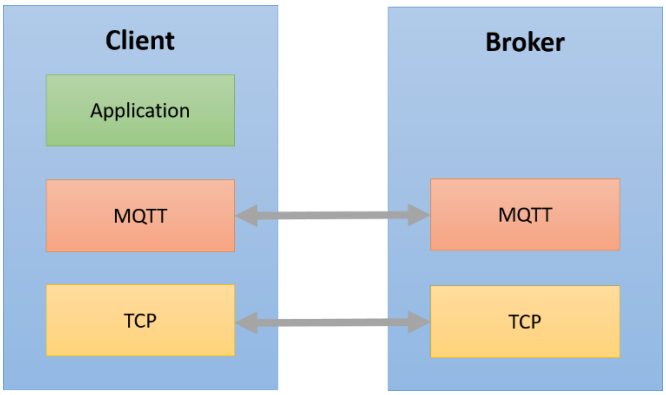
2.3.1.2. MQTT

Az MQTT (Message Queue Telemetry Transport) [11] egy OASIS szabvány, ami TCP/IP protokoll felett biztosítja a publish-subscribe minta szerinti kommunikációt. Az MQTT-t kifejezetten az erőforráskorlátos és alacsony sávszélességgel rendelkező eszközök (M2M) közötti kommunikációra fejlesztették ki. A használatához kevés memóriára van szükség, takarékosan bánik az adatcsomagokkal és hatékony adatszórást tesz lehetővé, ezzel biztosítja azt, hogy mobil eszközökben is jól használható legyen, és hosszú élettartamot nyújtson az akkumulátorok számára.

MQTT működés

Az MQTT terminológiájában két résztvevő fél különböztethető meg:

- **Client**: A program, ami MQTT-t használ. Kialakítja a kapcsolatot a brókerrel.
- **Broker**: Közvetítő szerepet játszik a Client-ek között. Az üzenetek az ő irányításukkal jutnak célba.



Ábra 2-12 - MQTT működése

A Client-ek a Broker-rel való kapcsolat felépítése után feliratkozhatnak **Topic**-okra vagy ők maguk publikálhatnak adatokat (**Application Message**) a feliratkozott Client-ek számára. A Topic-ról való leiratkozásra is van lehetőség.

A Broker kezeli az üzenetek továbbítását (fogadja a küldő Client-ek üzenetét és továbbítja az arra a Topic-ra feliratozott Client-ek számára), a fel- és leiratkozási kéréseket és a kezeli a Client-ekkel való kapcsolat kiépítését.

A szabvány, hasonlóan az ICMP protokollhoz, vezérlő üzeneteket (**MQTT Control Packet**) is definiál, olyanokat mint a Broker-hez való kapcsolódás (*CONNECT*) kérelme vagy a Topic-ról való leiratkozásé (*UNSUBSCRIBE*).

A Topic azonosítja az üzenetet, struktúrájuk hierarchiába szervezhető (pl.: ház/szoba1/világítás, ház/garázs/riasztó). Ezek a struktúrák egy-egy típusú üzenetet határoznak meg. A kliens feliratkozhat egy vagy akár több topikra is. Erre lehetőséget nyújtanak a helyettesítő (Wildcard) karakterek (#, +), amikkel szűrések határozhatók meg a topikokra (pl.: *home/#* = minden *home/*-el kezdődő topik). Egy kliens egyszerre csak egy topikba publikálhat üzeneteket, így ebben az esetben nem használhatók a helyettesítő karakterek.

Az üzenetek felruházhatók QoS szabályokkal. Az MQTT három szabályt definiál:

- **(0) At most once** – Best effort, az üzenetek elveszhetnek, nincs garancia a célba érkezésre
- **(1) At least once** – Garantálja az üzenetek eljutását a célba, legalább egyszer
- **(2) Exactly once** – Lényeges, hogy az alkalmazás pontosan egyszer kapja meg az üzenetet

Ha egy fogadó kliens offline, a QoS 1 és 2 üzenetek egy sorban tárolódnak, ahonnan az újra elérhető váló kliens megkaphatja az adatokat.

2.3.1.3. Blockchain

Milyen lehetőségünk van akkor, ha a rendszerünkben előfordulnak olyan kritikus adataink, amiknek a letagadhatatlansága létfontosságú? Letagadhatatlanság az, amikor egy tranzakcióban szeretnénk, hogy a tranzakcióban résztvevő adatokat egyik résztvevő fél se tagadhassa le, egyöntetűen elfogadják a tranzakció valódiságát és létrejöttét. Különböző tranzakciók vagy a naplózás letagadhatatlansága fontos lehet az integritás szempontjából.

A letagadhatatlanság biztosításához használhatóak digitális aláírások, publikus kulcsú titkosítási megoldások is. Ezen módszerek használatához általában szükség van egy *megbízható harmadik fél (TTP = Trusted third party)* bevonására. Azonban harmadik fél bevonása nélkül is lehetséges a letagadhatatlanság biztosítása és a *partnerkockázat* (*= a befektetések adásvétele, őrzése, kezelése és értékelése során külső szolgáltatók és partnerek alkalmazásából adódó kockázatok*) megszüntetése, emellett a tranzakciós költségek is csökkenhetnek.

A Blockchain, olyan, mint egy publikus főkönyv, ahol a tranzakciók (tágabban blokkok) listaként, kriptográfia eljárások által kapcsolódnak össze, egy láncot alkotva. A listába csak a résztvevők által elfogadott tranzakciók kerülhetnek be. Minden blokk tartalmaz egy mutatót az előző és a következő blokkra. Tervezésnél fogva ez a struktúra ellenáll az adatok módosításának, ezzel biztosítva a letagadhatatlanságot.

Léteznek publikus (Etherium) és privát (Hyperledger Fabric) blockchain megoldások. A privát blockchain előnye a publikussal szemben az, hogy korlátozni lehet a hozzáférést a rendszerhez, így egyfajta felügyeletet biztosítva (pl.: autóipari beszállítói lánc).

Ez a technológia is felhasználható CPS-ben, kielégítve az erre vonatkozó követelményeket.

2.3.2. Céltechnológiák összehasonlítása

Egy éles rendszerben a választás az MQTT és a DDS meggondolandó. A választás nagyban függ a feladattól és annak kontextusától, univerzális megoldás nem létezik. Mind a két megoldás támogatja az IoT-ben elvárt szolgáltatásokat.

A küldő és a fogadó felek közti virtuális összeköttetést mind a két esetben a topikok, azaz maga az adat adja. Az interoperabilitás mind a két esetben biztosítva van a rendszerek és a komponensek között.

Az MQTT és a DDS előnye a többi tárgyalt megoldással szemben, hogy alkalmazhatók egészen az Edge-től a Cloud-ig tartó kommunikációban.

Az MQTT és a DDS közti alapvető különbség az üzenetek továbbításának módjában van.

- Az MQTT egy centralizált, bróker alapú üzenetküldő szolgáltatást valósít meg, ahol a bróker feladata a kapcsolat kialakítása, az üzenetek továbbítása és a QoS szolgáltatások biztosítása.

- A DDS teljesen decentralizált peer-to-peer megoldáson alapuló üzenetküldést tesz lehetővé. Alapjául a korábban ismertetett LDA szolgál. A DDS magában, az őt használó eszközben cashelési technikákkal biztosítja a működést (üzenetek sorba rendezése, szűrése, tárolása).

A két megoldás összehasonlításának alapjául az erőforrásfelhasználásuk (memória, processzor, hálózati erőforrások) és a legkülönfélébb eszközök támogatása (mobil, beágyazott, valós idejű operációs rendszerek) szolgálhat [31].

Az alábbi feltételek alapján kiválasztható a megfelelő technológia.

	MQTT	DDS
Hálózat	WAN, LAN	osztott memória, busz, LAN, WAN
Üzenetek száma	Néhány üzenet/s eszközönként	Akár 10000 üzenet/mp eszközönként
Késleltetés	Néhány 100ms	Néhány 100 µs
Valós idejű működés	Amennyire lehetséges	Teljes támogatás
Hibatűrés	Rövid szolgáltatáskiesés megengedett	Szolgáltatáskiesés katasztrófális lehet
Interoperabilitás	Prtotokoll szinten	Protokoll és alkalmazás szinten

Táblázat 2-2 - MQTT és DDS összehasonlítása

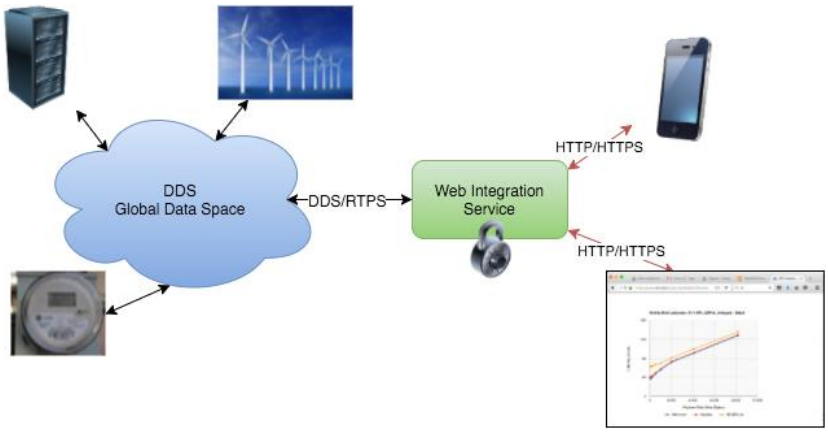
Valós idejű rendszerekben az MQTT használata nem javasolt, általánosságban nem támogatja az ezekre vonatkozó követelményeket, míg a DDS a rendkívül alacsony késleltetésével és a rengetek QoS szolgáltatásával ilyen rendszerekben is megfelelő.

Egy összetett rendszerben mind a kettő megoldás használható. A rendszerek közti átjárást egy *Gateway* (= komponens, aminek segítségével különböző típusú hálózatokat lehet összekapcsolni) segítségével könnyen ki lehet küszöbölni. Az MQTT leginkább a rendszer telemetria alkalmazásaiban használható, míg a kritikus valós idejű megoldásokban a DDS használata biztosabb.

2.3.3. Heterogén megoldások

CPS-ben gyakran előfordul, hogy a követelmények kielégítéséhez több különböző technológia alkalmazására is szükség van. Általában a technológiákon belül az interoperabilitás megoldott, de a technológiák között ez nincs biztosítva.

A megoldások közti inteoperabilitást a vendorok gyakran a saját technológiájukhoz készített Gatewayek segítségével biztosítják.



Ábra 2-13 - DDS Web Integration Service Gateway[9]

Például az RTI Web Integration Service[8] egy Gatewayt valósít meg a DDS és a webes szolgáltatások között. Felhasználásával könnyen készíthetők webalkalmazások a DDS-ből származó adatok felhasználásával.

A Gateway elvégzi azokat a konverziókat (pl.: üzenet formátumának módosítása), amikkel az üzenetek a rendszerek között is továbbíthatók lesznek.

A referenciaarchitektúrák bemutatásánál is látható volt, hogy a Gateway sokszor elengedhetetlen része a CPS architektúrának.

2.3.4. Technológiák felhasználása

A kiválasztott technológiáknak a tervezett funkcióra való alkalmasságának [1] [14] vizsgálatához egyszerre kell figyelembe venni jellemzőiken és működésükön túlmenően, az extra-funkcionális követelményeknek való megfelelőségüket. Ugyanakkor vannak olyan funkciók (pl.: letagadhatatlanság), amiket bizonyos technológiákban az alapelemek nem nyújtanak szolgáltatásként, de azokat más komponensekkel kiegészítve vagy megfelelő felhasználási mód mellett meg lehet valósítani (tervezési minták). A rendszer ellenőrzése során, e vizsgálat alapján megmondható, hogy a választott technológia valóban teljesíteni tudja-e a rá szabott követelményeket.

A táblázatból látszik, hogy vannak olyan követelményosztályok, melyeket alapesetben csak egy-egy technológia teljesít (pl.: letagadhatatlanság→Blockchain), ugyanakkor, ha ezekre a tulajdonságokra már figyelembe vesszük a tervezési mintákat is, akkor a fizikai jellegű korlatokon túl (pl.: átbocsátóképesség) a legtöbb esetben tervezési alternatívák vannak. Egy nagyobb rendszerben tehát egy induló terv esetében támogatást igényel önmagában a követelményeknek való megfelelőség ellenőrzése is. A tervezes folyamán előnyös, ha a részlegesen kidolgozott modell esetén meghatározhatók a további tervezési alternatívák (a tervezési tér).

Aspektusok támogatása		DDS	MQTT	Blockchain
Késleltetés	Alacsony	Támogatott	Nem támogatott	Nem támogatott
	Közepes	Támogatott	Támogatott	Nem támogatott
	Magas	Támogatott	Támogatott	Támogatott
Átersztőképesség	Alacsony	Támogatott	Támogatott	Támogatott
	Közepes	Támogatott	Támogatott	Nem támogatott
	Magas	Támogatott	Nem támogatott	Nem támogatott
Időbeliség		Támogatott	Nem támogatott	Nem támogatott
Letagadhatatlanság		Nem támogatott	Nem támogatott	Támogatott
Naplózás	Alap komponens	Nem támogatott	Nem támogatott	Támogatott
	Külső naplózási csomópont	Támogatott	Támogatott	
Szűrés	Alap komponens	Támogatott	Nem támogatott	Nem támogatott
	Topic szinten		Támogatott	
Megbízhatóság	Alap komponens	Támogatott	Nem támogatott	?
	QoS Szolgáltatás		Támogatott	
Futás idejű konfiguráció	Alap komponens	Támogatott	Nem támogatott	Nem támogatott
	Fejlesztési cél			Támogatott

Táblázat 2-3 - Technológiák által támogatott aspektusok

A táblázat a három legelterjedtebb elosztott rendszerek integrációjára szolgáló megoldást hasonlít össze az alapvető extra-funkcionális követelmények szempontjából. A táblázatban az egyes mennyiségek jellemzőit kvalitatív kategóriákra osztottam, külön feltüntettem az egyszerű általános lokális tervezési

[8] https://community.rti.com/static/documentation/connext-dds/5.2.3/doc/manuals/web_integration_service/index.html

mintákat, mint például egy csomóponthoz naplózás csatolása (architektúrális minta), illetve a megbízhatóság esetében az üzenet nyugtázása ("tértivevény"), mint funkcionális minta. Természetesen a későbbiekben alkalmazandó megközelítés során törekedtem arra, hogy a szokványos tervezés esetén az ilyen stílusú tervezési minták befogadhatók legyenek. A Blockchain esetében a jelenlegi, még korai implementációk egyes hibáinak kiküszöbölésen dolgoznak (Linux Foundation Hyperledger Performance Working Group). {Kocsis Imre, munkabizottsági tag személyes közlése.}

3. Szolgáltatásbiztonság ellenőrzése

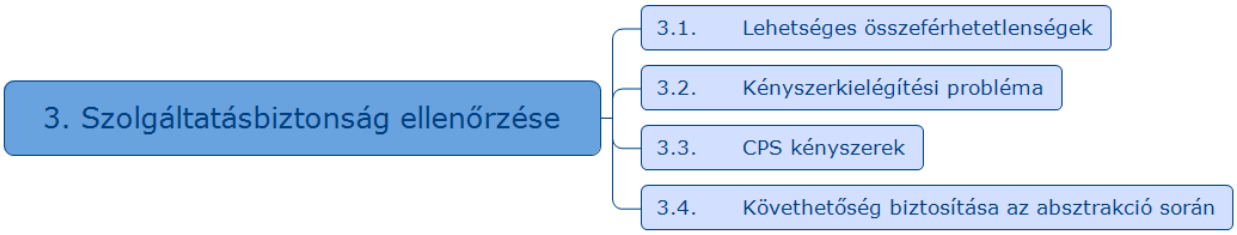
A CPS szolgáltatásbiztonságának ellenőrzése elengedhetetlen a rendszer helyes működésének garantálásához. A tervezési fázisban való ellenőrzés a fejlesztés korai szakaszában meg tudja mutatni a követelmények és a megvalósításhoz választott technológiák által előidézett összeférhetetlenségeket.

A cél egy módszertan meghatározása volt, amivel a rendszer architektúrájának és a követelmények tudatában automatikusan feltárhatók a felmerülő hibák, melyek javítása a tervezés szakaszában megtörténhet.

A fejezetben bemutatásra kerül, hogy a tervezés során a követelményekből fakadóan milyen hibák léphetnek fel a kész (implementált) rendszerre nézve.

Ismertetésre kerül az a kényszerkielégítési probléma és annak felhasználási módja az automatikus ellenőrzés biztosítására.

Végül szemléltetve lesznek az eszközök (modellező, megoldó), amik segítségével ez a módszer elvégezhető.



Ábra 3-1 - Szolgáltatásbiztonság ellenőrzése - felépítés

3.1. Lehetséges összeférhetetlenségek

A CPS rendszer tervezése során a komponensekhez rendelt követelmények ellentmondó szolgáltatásigényeket határozhatnak meg. A felhasználói igények követelményekké alakítása során az egyes keretrendszerek (pl.: NIST CPS Framework) nyújtotta módszerek használata nem garantálja az esetleges ellentmondások felfedezését. Ezen felül előfordulhat, hogy az egyik komponens által használt technológia sem tudja biztosítani a többi rá- vagy a kapcsolódó komponensekre vonatkozó igényeket. Ez az ellentmondás tervezés során nem mindig vehető észre egyértelműen. Szükséges lehet ennek ellenőrzése.

A kényszerek szisztematikus ellenőrzésével egy kis rendszerben kézzel könnyen feltárhatók az összeférhetetlenségek. Az ellenőrzéshez megfelelő alapul szolgálhat a *2-3* táblázat felépítése, amiből megállapítható, hogy a komponensek követelményei és a követelmény megvalósításához választott

technológia milyen viszonyban vannak egymással, azaz, hogy az adott követelményekhez választott technológia használható-e.

Ezen korlátozások formalizálásával felírható a rendszerre egy olyan általános érvényű szabályrendszer, ami alapján automatikusan ellenőrizhető az összes komponens megfelelősége, a fent említett kényszerekre vonatkozóan.

Ezek a kényszerek a csomópontok egyes követelményei között lépnek fel, így bizonyos tulajdonságok meghatározásával (pl.: technológia megválasztása), korlátozódik az extra-funkcionális követelményekre vonatkozó szabadság (pl.: a technológia kiválasztása meghatározza, a csomópont átbocsátóképességére vonatkozó tulajdonosát).

Ezekkel a kényszerekkel kiegészített szisztematikus ellenőrzéssel automatizálható a szolgáltatásbiztonság ellenőrzése, azaz egy kényszerkielégítési probléma megoldásával az ellenőrzés végrehajtható.

3.2. Kényszerkielégítési probléma

A kényszerkielégítési probléma [2] [3] [19] (CSP = Constraint Satisfaction Problem) egy matematikai probléma, amit formálisan változók (*variables*) , kényszerek (*constraints*), és a változók lehetséges értékeinek (tartomány) (*domains of values*) halmazaként lehet megadni:

$< \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{C} >$ egy kényszerkielégítési probléma, ahol

- $\boldsymbol{X} = \{\boldsymbol{x_1}, \dots \boldsymbol{x_n}\}$ – Változók halmaza.
- $\boldsymbol{D} = \{\boldsymbol{D_1}, \dots \boldsymbol{D_m}\}$ – Változók lehetséges értékeinek halmaza (változó értékkészlete).
- $\boldsymbol{C} = \{\boldsymbol{C_1}, \dots \boldsymbol{C_k}\}$ – Kényszerek halmaza.

Minden $\boldsymbol{X_i}$ változó értékeket vehet fel a nemüres $\boldsymbol{D_j}$ tartományból. Minden egyes $\boldsymbol{C_i} \in \boldsymbol{C}$ kényszer meghatározza egy adott $\boldsymbol{x} \subseteq \boldsymbol{X}$ halmazon a megengedett értékkombinációkat (relációkat).

Egy CSP **hozzárendelés konzisztens** és **teljes**, ha

$$\boldsymbol{S} = \{ \forall \boldsymbol{x_i} \in \boldsymbol{X} \rightarrow \boldsymbol{v_i} \in \boldsymbol{D_{x_i}} : < \boldsymbol{x_i}, \boldsymbol{v_i} > \subseteq \boldsymbol{C} \}$$

azaz, ha minden változóhoz értéket rendelünk a hozzá tartozó tartományból és ezek a hozzárendelések nem sértenek meg egy kényszert sem.

Ezek a konzisztens és teljes megoldások adják a megoldást/megoldásokat.

A változók tartományai lehetnek diszkrétek és folytonosak. A feladat megoldható diszkrét változók használatával, ezért a további állítások a diszkrét kényszerkielégítési problémára igazak.

$\boldsymbol{Z}$ célfüggvény:

$$\boldsymbol{Z = \sum_{i=1}^{n} c_i X_i}$$

,ahol
$\boldsymbol{c_i}$ = a célfüggvény együtthatója az $\boldsymbol{i}$. Változóra nézve, és
$\boldsymbol{X_i}$ = az $\boldsymbol{i}$. vizsgált változó

$\boldsymbol{M}$ megoldáshalmaz lehet:

- $\boldsymbol{M = \{S_1\} : S_1 \in S}$, azaz egy megoldást ad.
- $\boldsymbol{M = \{S_1, S_2, \dots, S_n\} : \cup S_i = S}$, azaz az összes megoldás.
- $\boldsymbol{M = opt(S_1, S_2, \dots, S_n) : S_i \in S \wedge opt()}$, ahol $\boldsymbol{opt()}$ célfüggvény a változók összességére vagy részére vonatkozóan. Ez egy célfüggvény szerinti optimális megoldást ad.

3.2.1. Kényszerek kategorizálása

A kényszerek három csoportba sorolhatók:

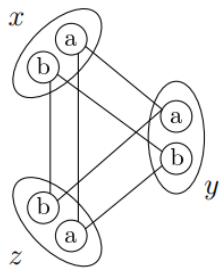
- **Unáris** kényszer – Csak egy változóra vonatkozó kényszerek
- **Bináris** kényszer – Két változóra vonatkozó kényszerek
- **Magasabb rendű** kényszerek – Több változóra vonatkozó kényszerek.

3.2.1.1. Magasabb rendű kényszerek

Egy **gráf** egy rendezett pár, $\boldsymbol{G = (V, E)}$, ahol $\boldsymbol{V}$ egy nem-üres halmaz, $\boldsymbol{E}$ pedig ebből a halmazból képezhető párok egy halmaza. $\boldsymbol{V}$ elemeit pontoknak vagy csúcsoknak, $\boldsymbol{E}$ elemeit éleknek nevezzük.

Ha egy $\boldsymbol{G}$ gráfról beszélünk, akkor $\boldsymbol{V(G)}$-vel illetve $\boldsymbol{E(G)}$-vel jelöljük a gráf csúcsainak illetve éleinek halmazát, míg a pontok illetve élek számát $\boldsymbol{v(G)}$-vel ill. $\boldsymbol{e(G)}$-vel jelöljük.

Ha az $\boldsymbol{e} \in \boldsymbol{E}$ él a $\{\boldsymbol{v_1}, \boldsymbol{v_2}\}$ párnak felel meg, akkor ez a két pont $\boldsymbol{e}$ végpontja. Ha $\boldsymbol{v_1 = v_2}$, akkor $\boldsymbol{e}$ **hurokél**. Ha két különböző nem hurokélnek a végpontjai azonosak, a két élet párhuzamos vagy többszörös élnek nevezzük. Azokat a gráfokat, amelyekben nincsenek hurokélek és többszörös élek, egyszerű gráfnak nevezzük.



Ábra 3-2 - Kényszergráf példa

A magasabb rendű kényszereket **kényszerhipergráffal** (constraint hyprergraph) lehet ábrázolni.

> $\boldsymbol{H}$ **hipergráf**,
>
> $\mathbf{H = <X, E>}$
> $\mathbf{X = (x_i \mid i \epsilon I_v)}$, a gráf csomópontjai (hipercsúcsok).
> $\mathbf{E = (e_i \mid i \epsilon I_e, e_i \subseteq X)}$, a gráf élei (hiperélek), a csomópontok részhalmazai alkotják.

> $\boldsymbol{H}$ **kényszerhipergráf**,
>
> Ha $\boldsymbol{P} = < \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{C} >$ kényszerkielégítési probléma és $\boldsymbol{H} = < \boldsymbol{X}, \boldsymbol{E} >$ hipergráf
>
> - $\boldsymbol{H}(\boldsymbol{X}) \rightarrow \boldsymbol{P}(\boldsymbol{X})$*, a hipercsúcsok a változókat jelölik*
> - $\boldsymbol{H}(\boldsymbol{E}) \rightarrow \boldsymbol{P}(\boldsymbol{C})$*, a hiperélek pedig a kényszereket.*
>
> Minden kényszerben résztvevő változó eleme a hiperélnek.

Kényszerek binarizálása

A kényszerhipergráf egy hipergráf, ami ábrázolható egyszerű gráfként, ahol:

> 1. $\boldsymbol{v_i} \in \boldsymbol{V} \rightarrow (\boldsymbol{x_i} \in \boldsymbol{X} \vee \boldsymbol{c_i} \in \boldsymbol{C})$, azaz a csomópontok lehetnek változók vagy kényszerek.
> 2. Ha az $\boldsymbol{e_i} \in \boldsymbol{E}$ él a $\{\boldsymbol{v_1}, \boldsymbol{v_2}\}$ csomópontpárnak felel meg, akkor $\boldsymbol{v_1} \in \boldsymbol{X} \wedge \boldsymbol{v_2} \in \boldsymbol{C}$, azaz egy él csak egy változó-csomópontot és egy kényszer-csomópontot köthet össze.
> 3. Az $\boldsymbol{e_i} \in \boldsymbol{E}$ él a $\{\boldsymbol{v_1}, \boldsymbol{v_2}\}$ csomópontok között csak akkor futhat, ha $\boldsymbol{v_1} \in \boldsymbol{X} \wedge \boldsymbol{v_2} \in \boldsymbol{C}$, és $\boldsymbol{v_1} \in \boldsymbol{v_2}$, azaz $\boldsymbol{v_1}$ változó részt vesz a kényszerben.

Az 1. és a 2. tulajdonságokból következik, hogy a kényszergráf **páros gráf**.

> Egy $\boldsymbol{G}$ gráfot páros gráfnak nevezünk, ha a $\boldsymbol{G}$ pontjainak $\boldsymbol{V}(\boldsymbol{G})$ halmaza két részre, egy $\boldsymbol{A}$ és $\boldsymbol{B}$ halmazra osztható úgy, hogy $\boldsymbol{G}$ minden élének egyik végpontja $\boldsymbol{A}$-ban, másik végpontja $\boldsymbol{B}$-ben van. Ennek jelölése: $\boldsymbol{G} = (\boldsymbol{A}, \boldsymbol{B})$.

A megfelelő számú segédváltozó (kényszer-csomópont) hozzáadásával a magasabb rendű kényszer felírható bináris kényszerek halmazával.

A probléma egyszerű gráfként való ábrázolása segítséget nyújt a megoldás szempontjából. A gyakorlatban gráfalgoritmusokat használnak a feladatok megoldásához.

3.2.1.2. Unáris és bináris kényszerek

A legtöbb CSP algoritmus olyan problémákra alkalmazható, ahol a kényszerek **unárisak** vagy **binárisak**.

Unáris kényszer esetén a változó értékére tehető megkötés (pl.: az adatok felhőbe való eljuttatásának késleltetése bizonyos küszöbérték felett lesz). Ezek az unáris kényszerek megszüntethetők a változó tartományának lecsökkentésével, azaz a változó tartományában csak a megengedett értékek maradnak bent (pl.: csak a küszöbérték fölötti késleltetés a megengedett).

A csak bináris kényszereket tartalmazó CSP a **bináris CSP**. A bináris CPS ábrázolható gráfként.

A **kényszergráf** (constraint graph) olyan gráf, ahol a csomópont a kényszerkielégítési probléma változóit, az élek pedig a kényszereket jelölik, azaz

$\forall \boldsymbol{v_i} \in \boldsymbol{V} \rightarrow \boldsymbol{x_i} \in \boldsymbol{X}$,és

$\forall \boldsymbol{e_k} \in \boldsymbol{E} \rightarrow \boldsymbol{c_k} \subseteq \boldsymbol{C}$

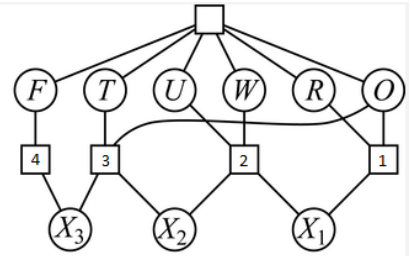
CSP példa

Az alábbi ábrán egy betűrejtvény látható. A rejtvényben az összeadáskényszerek felírhatók a következő egyenletekkel:

$$(1)\ O + O = R + 10 \cdot X_1$$
$$(2)\ X_1 + W + W = U + 10 \cdot X_2$$
$$(3)\ X_2 + T + T = O + 10 \cdot X_3$$
$$(4)\ X_3 = F$$

Az $\boldsymbol{X_1, X_2, X_3}$ változók a következő oszlopba átvitt számjegyeket jelölik, ezek a segédváltozók segítenek a feladat megoldásába. Az ábra jobb oldalán a feladat hipergráfként való ábrázolása látható. Az ábra harmadik sorában lévő kényszercsomópontokhoz azok a változók kapcsolódnak, amik az adott egyenletben részt vesznek.

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$



Ábra 3-3 - Betűrejtvény példa

3.2.1.3. Preferencia kényszerek

Az **abszolút kényszer**eken (amiknek a megsértése kizár egy megoldásjelöltet) kívül léteznek **preferencia kényszer**ek is. A preferenciakényszerek megmutatják melyik megoldások preferáltak (pl.: autóvásárlás során a vevő a piros színt részesíti előnybe). A preferenciakényszerek szerepe az optimális megoldás megtalálása során kerül elő, optimálisabbnak tekinthető a megoldás, ha a változó a preferált értéket veszi fel (pl.: a megoldáshalmazban lévő piros és kék autók közül a piros lesz az optimálisabb).

3.2.2. CSP megoldása

A CSP megoldása során az első lépés az előfeldolgozás (**preprecessing**). A folyamat feladata az unáris kényszerek vizsgálata a csomópontokon. Ezekhez a vizsgálatokhoz, csak az adott csomópont vizsgálatára van szükség. Az előfeldolgozási lépés végrehajtásával lényegesen lecsökkenthető az egyes változókhoz tartozó tartományok nagysága. A tartományok lecsökkentésével a CSP megoldását célzó fázisok

hatékonysága és gyorsasága sokkal jobb lehet (pl.: bináris típusú változó esetén, a megoldástér fele kizárásra kerülhet). A nagyméretű problémák megoldásánál ez a lépés elengedhetetlen.

A CSP megoldására az egyik módszer a generálás-és-tesztelés (**generate-and-test**), ami szisztematikusan legenerálja az összes lehetséges értékhozzárendelést, majd ellenőrzi, hogy teljesült-e az összes kényszer az adott hozzárendelésre. A módszer nem túl hatékony, mivel nagyon sok hibás hozzárendelést is legenerál, amik a tesztelés fázisában eldobásra kerülnek.

Ennél hatékonyabb megoldást lehetne elérni, ha egy érték hozzárendelése után rögtön ellenőrizni lehetne a kényszerek teljesülését. Ezen az elven alapszik a visszalépéses (**backtrack**) módszer. A módszer inkrementálisan próbál kiegészíteni egy részleges megoldást úgy, hogy a további változók értékének hozzárendelésével se sérüljön a konzisztencia. Ha egy változóhoz nem található érték, úgy hogy azzal ne sérüljenek a kényszerek, akkor egy korábbi változó értékét kell módosítani (vissza kell lépni), így a későbbiekben egy konzisztens és teljes megoldás kialakulhat.

Mivel a backtrack algoritmusnak nincs információja a megoldás állapotáról (célállapotok megkülönböztetése), így nem tudja megmondani, hogy a konfliktus milyen okból jött létre, ezért többször bele tud futni hasonló hibába, ezzel redundáns munkát is végez. Az algoritmus javítására (a keresési tér méretének csökkentésével) több módszer is használatos:

- Előrenéző ellenőrzés (forward checking)
- Kényszerek terjesztése

Előrenéző ellenőrzés: Minden olyan esetben, amikor egy $\boldsymbol{X}$ változó értéket kap, a hozzá kényszerrel kapcsolt, de még nem meghatározott $\boldsymbol{Y}$ változó tartományából törli az $\boldsymbol{X}$ változó értékével nem kompatibilis értékeket.

```
procedure NC
  for each V in nodes(G)
    for each X in the domain D of V
      if any unary constraint on V is inconsistent with X
      then
        delete X from D;
      endif
    endfor
  endfor
end NC
```

Kényszerek kiterjesztése: Alapja, hogy az egyik változó kényszerének a többi változót érintő következményei terjednek a rendszerben.

Bináris kényszergráfban meg lehet győződni az egyes élek konzisztenciájáról (élkonzisztencia – arc consistency), azaz arról, hogy a csomópontok kielégítik-e az adott élre vonatkozó kényszert.

Élkonzisztencia vizsgálata

A $(\boldsymbol{V_i}, \boldsymbol{V_j})$ él **élkonzisztens**, ha a $\boldsymbol{V_i}$ tartományában lévő minden $\boldsymbol{x}$ értékekhez létezik a $\boldsymbol{V_j}$ tartományában lévő olyan $\boldsymbol{y}$ érték, amire $\boldsymbol{V_i = x, V_j = y}$ megengedett kényszer.

Egy $(\boldsymbol{V_i}, \boldsymbol{V_j})$ él konzisztensé tehető, ha a $\boldsymbol{V_i}$ tartományából kitörlésre kerül minden olyan érték, ami nem konzisztens a $\boldsymbol{V_j}$ tartományában található valamelyik értékkel. Ezzel a módszerrel a $(\boldsymbol{V_i}, \boldsymbol{V_j})$ csúcsok közti kényszer kielégítésre kerül. A következő algoritmus ezt a folyamatot hajtja végre.

```
procedure REVISE(Vi,Vj)
  DELETE <- false;
  for each X in Di do
    if there is no such Y in Dj such that (X,Y) is consistent,
    then
       delete X from Di;
       DELETE <- true;
    endif;
  endfor;
  return DELETE;
end REVISE
```

Ahhoz, hogy az összes él élkonzisztens legyen, a fenti algoritmust nem elég egyszer lefuttatni minden csúcson. A $\boldsymbol{V_i}$ csúcs felülvizsgálata során lehetnek olyan $(\boldsymbol{V_i}, \boldsymbol{V_j})$ élek, ahol a $\boldsymbol{V_i}$-ből való törlés miatt megszűnik a konzisztencia.

Az *AC-1* algoritmus addig ellenőrzi az éleket, amíg lesz olyan csúcspont, aminek a tartományát szűkítheti. Ez nem hatékony, mivel sokszor megvizsgálja azokat az éleket is, amik nem kapcsolódnak az éppen felülvizsgált csúcshoz.

Az *AC-3* algoritmus eltárolja, hogy melyik éleket kell felülvizsgálni. Minden $(\boldsymbol{V_i}, \boldsymbol{V_m})$ él felülvizsgálata után, a hozzá kapcsolódó $(\boldsymbol{V_i}, \boldsymbol{V_j}), ahol\ \boldsymbol{j \neq m}$ éleket újra fel kell venni a vizsgálandó élekhez. Így csak azok az élek kerülnek újra felülvizsgálatra, aminek az egyik csúcsában a tartomány változott (egy része törlésre került). Az vizsgálatokat addig kell folytatni, amíg előfordul felülvizsgálandó él.

```
procedure AC-3
  Q <- {(Vi,Vj) in arcs(G),i#j};
  while not Q empty
    select and delete any arc (Vk,Vm) from Q;
    if REVISE(Vk,Vm) then
      Q <- Q union {(Vi,Vk) such that (Vi,Vk) in arcs(G),i#k,i#m}
    endif
  endwhile
end AC-3
```

Az algoritmus elvégzése után mindegyik él konzisztens lesz vagy egyes változók tartománya üres is lehet. Üres tartomány esetén a CSP nem megoldható.

3.2.3. XCSP3

Az XCSP3 [21] egy XML alapú formátum, ami kombinatorikus kényszer problémák leírására alkalmas. Meghatározza a probléma modelljét, kompakt, könnyen olvasható és programok által feldolgozható módon. Használható kényszerkielégítési és kényszeroptimalizációs feladatokra is. A meghatározott eszközkészletét (pl.: Meta-Constraints, Soft Constraints) kombinálva lehetőség van összetett problémák leírására is.

Az XCSP3 által támogatott adattípusok a következők:

- Logikai változók (0/1)
- Egész értékű változók
- Szimbolikus változók
- Valós számok
- Halmazok
- Sztochasztikus változók
- Kvalitatív változók
- Változók tömbjei

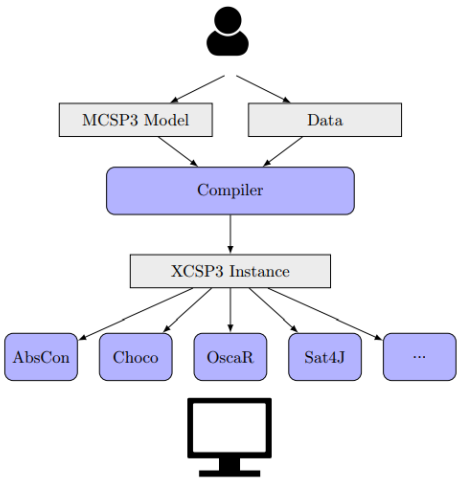
A támogatott kényszerekkel lehetőség van az összes adattípusra kényszereket meghatározni.

A formátum összetett feladatok leírására is biztosítja a megfelelő szintaktikát. Az összetartozó kényszerek osztályokba rendezhetők, így megkönnyítve az átláthatóságot és a szerkeszthetőséget.

XCSP3 biztosít egy Java könyvtárat [22] 3 eszközzel:

- **XCSP3-Java-Parser** – Feldolgozó könyvtár Java8 számára.
- **XCSP3-Java-Checker** – XCSP3-ban leírt problémák megoldásának ellenőrzésére használható eszköz.
- **MCSP3 (modeler)** – API-t biztosít modellek készítésére, amik XCSP3 fájlokra fordulnak.

A probléma modellezésétől a megoldásig vezető utat az alábbi ábra szemlélteti.



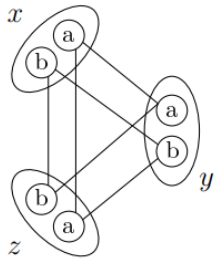
Ábra 3-4 - CSP megoldása XCSP3 használatával (átvéve: [21]-ból)

- MCSP3 használatával a modell elkészítése
- Példányok adatainak megadása
- A modell és az adat fordítása XCSP3 példánnyá
- Feladat megoldása (AbsCon, Choco, OscarR, Sat4J, stb. használatával)

Az XCSP3 példányban felírt CSP megoldható különböző programok (megoldók) használatával.

3.2.3.1. XCSP3 Példa

A következő példán jól látható az XCSP3 felépítése.



$< \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{C} >$ egy kényszerkielégítési probléma, ahol

- $\boldsymbol{X} = \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$
- $\boldsymbol{D} = \{\boldsymbol{a}, \boldsymbol{b}\}$, minden változóhoz.
- $\boldsymbol{C} = \{< \boldsymbol{x}, \boldsymbol{y} > \in \{(\boldsymbol{a}, \boldsymbol{a}), (\boldsymbol{b}, \boldsymbol{b})\}, < \boldsymbol{x}, \boldsymbol{z} > \in \{(\boldsymbol{a}, \boldsymbol{a}), (\boldsymbol{b}, \boldsymbol{b})\}, < \boldsymbol{y}, \boldsymbol{z} > \in \{(\boldsymbol{a}, \boldsymbol{b}), (\boldsymbol{b}, \boldsymbol{a})\}$

Az ebből felírt XCSP3 formátum a következő:

```
<instance format="XCSP3" type="CSP">
   <variables>
      <var id="x" type ="symbolic"> a b </var>
      <var id="y" type ="symbolic"> a b </var>
      <var id="z" type ="symbolic"> a b </var>
   </variables>
   <constraints>
      <extension>
         <list> x y </list>
         <supports> (a,a) (b,b) </supports>
      </extension>
      <extension>
         <list> x z </list>
         <supports> (a,a) (b,b) </supports>
      </extension>
      <extension>
         <list> y z </list>
         <supports> (a,b) (b,a) </supports>
      </extension>
   </constraints>
</instance>
```

Az XCSP3 fájl az `<instance>` nyitó és záró címkékkel kezdődik és fejeződik be. Közötte helyezkednek el a kényszerváltozók és a változókra vonatkozó kényszerek.

A példában az $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ változók szimbolikus típusúak és az értékkészletük $\{\boldsymbol{a}, \boldsymbol{b}\}$**.** Egyszerű változót a `<var>` tagokkal kell megadni. Attribútumként az azonosítóját és a típusát, míg a címkék között pedig a változók értékkészletét.

A kényszereket a <constraints> címkék között kell felvenni. Szimbolikus típusú változók esetén a kényszerek megadása két részből áll. Meg kell adnunk a változók listáját, amire a kényszer vonatkozik és azt, hogy mik a megengedett <supports> vagy tiltott <conflicts> felvehető értékek a változók sorrendjében.

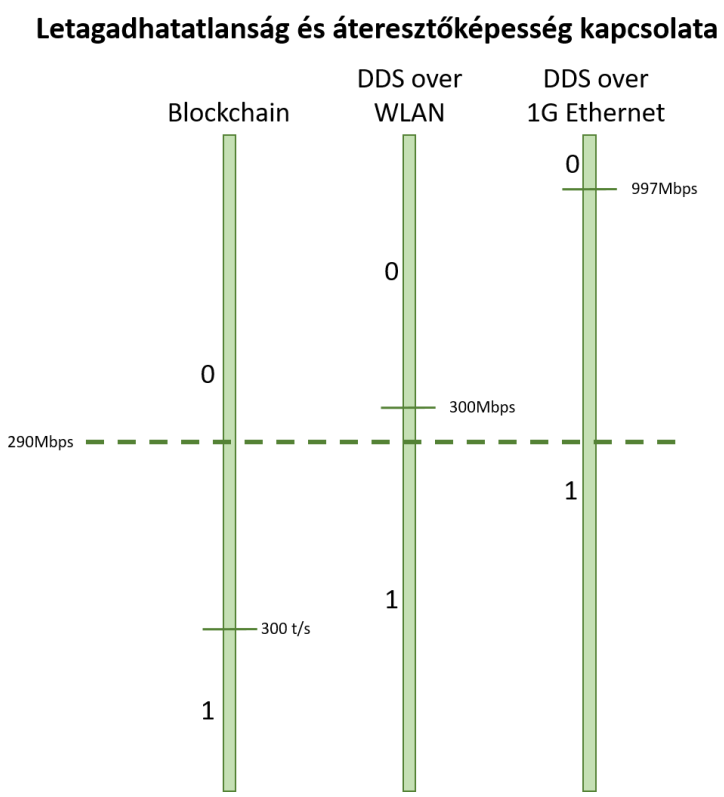
3.3. CPS kényszerek

3.3.1. Kényszerek kialakítása és finomítása

3.3.1.1. Kényszerek kialakítása

A CPS rendszerek tervezése során a komponensekre vonatkozó kényszerek felírásához a *2-3* táblázat megfelelő kiindulópont lehet. A kategóriák kialakítását bemutató példában használt módszerrel is meghatározhatók az egyes követelmények.

Kategóriák kialakítása



Ábra 3-5 - Technológia választása: Letagadhatatlanság és áteresztőképesség kapcsolata

A példában a letagadhatatlanság és az *áteresztőképesség* (= *megadja, hogy a rendszer hány kérést tud kiszolgálni időegység alatt*) kapcsolatának vizsgálata látható, három letagadhatatlanságot biztosító technológiát összehasonlítva (Blockchain, DDS over WLAN, DDS over 1G Ethernet). Az egyes technológiákra vonatkozó maximális áteresztőképességek az ábrán jelölve vannak. (pl.: A Blockchain maximum 300 tranzakciót tud végrehajtani másodpercenként.) Ezen határértékek alatt a választott technológia használható, viszont az ennél magasabb igényeket nem tudja kielégíteni. A követelményben meghatározott elvárt áteresztőképesség maximuma (worst case) az aktuálisan vizsgált csatornán 290Mbps. Az ábráról leolvasható, hogy mely technológiák tudják ezt az igény kielégíteni.

A technológiákra vonatkozó kifejezések ábrázolhatók egy-egy logikai formulaként (pl.: Blockchain esetén, $(\boldsymbol{K_{throughput} > L_{Blockchain}}) \rightarrow \{\mathbf{0}, \mathbf{1}\}, ahol\ \boldsymbol{K_{throughput}}$ az áteresztőképességre vonatkozó kényszer és $\boldsymbol{L_{Blockchain}}$ a Blockchain áteresztőképességére vonatkozó küszöbszint). A logikai formulák kiértékelésével megkapható, hogy melyek azok a technológiák, amik alkalmasak lehetnek a vizsgált követelmények kielégítésére.

Mivel ezek a kényszerek egy-egy csomópontra vonatkoznak csak, ezért az így meghatározott logikai kényszerek lehetőséget nyújtanak a CSP megoldása során az előfeldolgozási lépésben való tartománycsökkentéshez, biztosítva a tartományok szűrését a gyorsabb és hatékonyabb megoldás érdekében.

A példában az elvárt ($\boldsymbol{K_{throughput} = 290Mbps}$) és az egyes technológiák maximális ($\boldsymbol{L_{Blockchain} = 300\ tps, L_{DDSoWLAN} = 300Mbps, L_{DDSo1GEthernet} = 997Mbps}$) áteresztőképességéhez meghatározhatók a következő formulák:

- $\boldsymbol{(K_{throughput} > L_{Blockchain}) \rightarrow 0}$
- $\boldsymbol{(K_{throughput} > L_{DDSoWLAN}) \rightarrow 1}$
- $\boldsymbol{(K_{throughput} > L_{DDSo1GEthernet}) \rightarrow 1}$

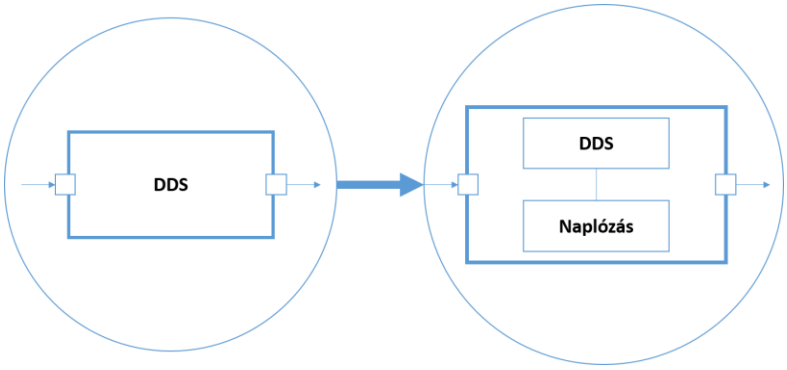
Leolvasható, hogy ezen kényszer kielégítésére a Blockchain nem megfelelő, így a CSP megoldása során a vizsgált komponenseknél nem jöhet szóba megoldásként.

Azonban ahogy az a *2-3* ábrán látható, a tervezési mintákkal kiegészített működésű csomópontok viselkedése eltérhet a várttól (pl.: DDS letagadhatatlansággal való kapcsolata), így szükség lehet az egyes kényszerek finomítására.

3.3.1.2. Kényszerek finomítása

A modellezésben gyakran használt módszer az egyes elemek részekre bontása, azaz a kompozicionalitás használata. Ez lehetővé teszi a rendszerek részrendszerekre való bontását úgy, hogy az eredeti rendszer érintkezési felületei a külvilág számára nem változnak. Ez a felbontás kihasználható a CPS tervezése és az ebből kialakuló kényszerhálók kialakítása során.

Az CPS-ben résztvevő komponensek kompozicionalitása miatt lehetőség van új funkcionalitásokkal kibővíteni azokat (pl.: DDS csomópont kiegészítése naplózó funkcióval), azonban ezeknek az új funkcióknak a hozzárendelése befolyással lehet az eredeti komponens tulajdonságaira. A befolyásolás mértéke függ az új funkcionalitást megvalósító részkomponens tulajdonságaitól.



Ábra 3-6 - Kompozicionalitás példa

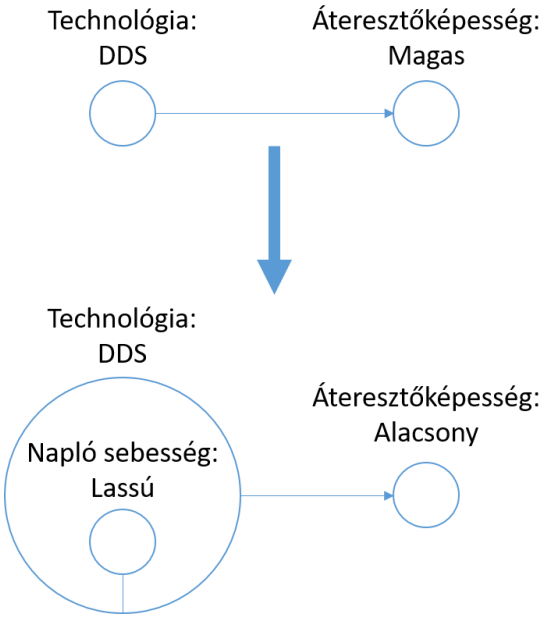
A példából látható, hogy az új funkcionalitásokat hozó részkomponens hogyan befolyásolhatja az eredeti komponens tulajdonságát.

Technológia		**DDS**		
Naplózás sebessége		-	Gyors	Lassú
Átersztőképesség	Alacsony	Támogatott	Támogatott	Támogatott
	Közepes	Támogatott	Támogatott	Nem támogatott
	Magas	Támogatott	Támogatott	Nem támogatott

A példán az önmagában használt DDS és a naplózással kiegészített architektúrális változatának összehasonlítása látható. Annak ellenére, hogy a DDS önmagában támogatja a magas áteresztőképességet, lassú naplózó komponens használatával ez nem biztosítható.

Ez a hierarchikus felépítés illeszkedik a CSP által meghatározott kényszerhálóban. Ahogy az architektúrális tervezés során az egyes csomópontok felbonthatók a funkcionalitásait megvalósító részcsomópontokra, úgy a kényszerhálóban lévő változók kompozicionalitási tulajdonágai is érvényesülnek.

Mivel a kényszerháló vizsgálatánál az egyes változókra is teljesül a kompozicionalitás, ezért lehetőség van a kényszerek finomítására úgy, hogy azzal az eredeti kényszerhaló felépítése ne módosuljon.



Ábra 3-7 - Kényszerhálók kompozicionalitása példa

Ezeket a tulajdonságokat kihasználva a kényszerek tetszőlegesen finomíthatók az igényeknek megfelelően, így összetett rendszerek ellenőrzésére is lehetőség nyílik.

3.3.2. CPS modellezése

A CPS tervezési folyamatban kritikus az architektúra tervezési fázis, hiszen ezek a rendszerek ma már jellegzetesen alrendszerek, komponensek és szolgáltatások integrációjával jönnek létre. Az architektúra tervezési fázis alapvetően megszabja az integrációs kereteket és kulcskérdés a funkcionális és extra-funkcionális követelmények mellett a valós idejű működéshez kapcsolódó időtartománybeli (pl.:

késleltetések, átbocsátóképesség, stb.), illetve a fizikai világ kapcsolódásához tartozó esetenként folytonos tartománybeli követelmények teljesítése.

Éppen a specifikus CPS tervezési aspektusok megfogalmazása és kezelése érdekében az OMG SysML (System Modeling Language) néven egy olyan CPS specifikus modellező nyelvet hozott létre a tisztán informatikai rendszereket kezelő UML bővítéseként, amely képes befogadni a fenti speciális aspektusokat is. A SysML-ben külön modellezési apparátus áll rendelkezésre a követelmények (requirement diagram) , a funkcionális és implementációs architektúra modellezésére.

Összetett CPS-ek esetén amennyiben valamilyen referenciaplatform felett készül a megoldás, e két architektúratervezési lépésben perdöntő a követelménynek való megfelelőség ellenőrzése, hiszen az itt fellépő specifikációs hibák akár az alap platform kiválasztását is érvényteleníthetik.

Munkámban ezért két kiválasztott referenciaplatformon illusztrálva, általános érvénnyel azt a kérdést tettem fel, hogy a SysML-ben [12] [17] megfogalmazott követelmények és egy funkcionális illetve implementációs architektúra konzisztenciáját hogyan lehet vizsgálni. A teljes részletességű, például értéktartományokban finoman definiáló modellek kezelése ilyenkor még nem cél, hiszen a kialakított architektúrát sokszor egy megoldáscsalád központi elemeként használják illetve a különböző futtatóplatformok között olyan jelentős különbségek vannak a szolgáltatásaikra vonatkozóan, hogy az ilyen felbontású vizsgálatnak közvetlen jelentősége nincs.

A fentiek alapján a formális vizsgálat alapja a követelmények és eszközök szolgáltatásainak kvalitatív modellezése. Egy adott protokoll szerinti együttműködést így néhány kategóriába lehet sorolni pl.: az átbocsátóképesség szempontjából (kis, közepes, nagy sebességű) ez az egyszerűsítés, amely tulajdonságokat paraméterintervallumokhoz rendel, egyrészt biztosítja a szabatos absztrakció után a helyességvizsgálat érvényességét, másrészt nyitott abban az értelembe, hogy speciális esetben a kategóriák definiálhatók.

A következőkben elsőként az absztrakciós mechanizmust [13] és a benne felhasznált eszközöket tárgyalom, majd röviden bemutatom azt a környezetet, amelyet a megközelítés tesztelésébe építettem.

3.3.2.1. CPS modell ábrázolása

3.3.2.1.1. SysML

A SysML egy általános célú modellező nyelv rendszertervezési felhasználásra. A rendszer vagy rendszerek egész életciklusára kiterjedő támogatást nyújt, a rendszer specifikációjától kezdve a rendszer *verifikáció*jának (*= objektív bizonyítékokkal való megerősítése a követelmények teljesítésének*) és *validáció*jának (*= külső követelményeknek való megfelelőség vizsgálata*) lépéséig. A SysML nyílt forrású OMG által felügyelt szabvány, amit az UML (Unified Modeling Language) egy részének kibővítéseként hoztak létre.

A SysML diagramból kiindulva lehetőség van a CPS ellenőrzésére a következő fejezetekben bemutatott módszerrel. A feladat megoldása során az egyik absztrakciós rétegben egy gráf alapú ábrázolást használtam. A SysML diagramon használt eszközök leképezhetők a gráf alapú ábrázolássá.

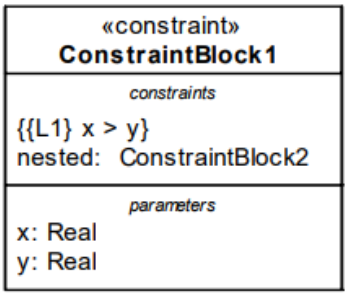
SysML eszköztár

A **Block** (blokk) egy moduláris elem, ami a rendszer vagy egy komponens struktúráját írhatja le. Tartalmazhat strukturális és a működési aspektusokból származó információkat. A blokkok rekurzívan további részekre bonthatók, ezzel modellezve a modularitást.

A **BDD** (Block Definition Diagram)-n definiálhatók a rendszerben használható komponensek (pl.: Sensor, Operator). CPS rendszerek tervezése során is használható. A CPS komponensei megtervezhetők a SysML eszközkészletével.

A modellben **Constraint** blokkok segítségével megadhatók kényszerek hálózatai matematikai kifejezésekkel. A kényszerek ábrázolhatók akár logikai kényszerként is, hozzákapcsolhatók több modellhez, amik megjeleníthetők különböző diagramokon. A követelmények követhetősége (traceability) által pontosan ismert, hogy melyik CPS komponens felelős az egyes megkötésekért.

A SysML szabvány szerint Constraint blokkot csak *BDD*-n vagy Package diagramon lehet definiálni, a <<constrain>> sztereotípia használatával. A *constraints* mezőben lévő kényszerek megfogalmazhatók informális nyelven vagy formálisan, OCL vagy MathML használatával is. A kényszerek paraméterei a *parameters* mezőben adhatók meg nevükkel és típusukkal együtt.



Ábra 3-8 - SysML ConstraintBlock

A fizikai kényszereken kívül a CPS szolgáltatásbiztonságra vonatkozó kényszerei is felvehetők.

Az egyes blokkok jellemezhetők a bennük felhasznált technológiával és a blokkra vonatkozó követelményekkel.

3.3.2.1.2. GraphML

A **GraphML** [4] [18] egy átfogó és könnyen használható XML fájlformátum gráfok leírása számára. A nyelv alapjaival meghatározható a gráf struktúrája és rugalmas bővítési lehetőséget biztosít az alkalmazásspecifikus attribútumok hozzárendelésére. A GraphML segítségével leírhatók:

- Irányított, irányítatlan és kevert gráfok
- Hipergráfok
- Hierarchikus gráfok

Lehetőség van:

- Attribútumok megadására
- Külső adatokra való hivatkozásra
- Alkalmazásspecifikus attribútumok (címkék) megadására

Ezeken felül könnyű feldolgozhatóságot, elemezhetőséget biztosít.

yEd Graph Editor

A feladat során használt GraphML szerkesztő az yWorks által elkészített yEd Graph Editor [23] alkalmazás volt. A grafikus szerkesztés során teljes GraphML támogatást nyújt:

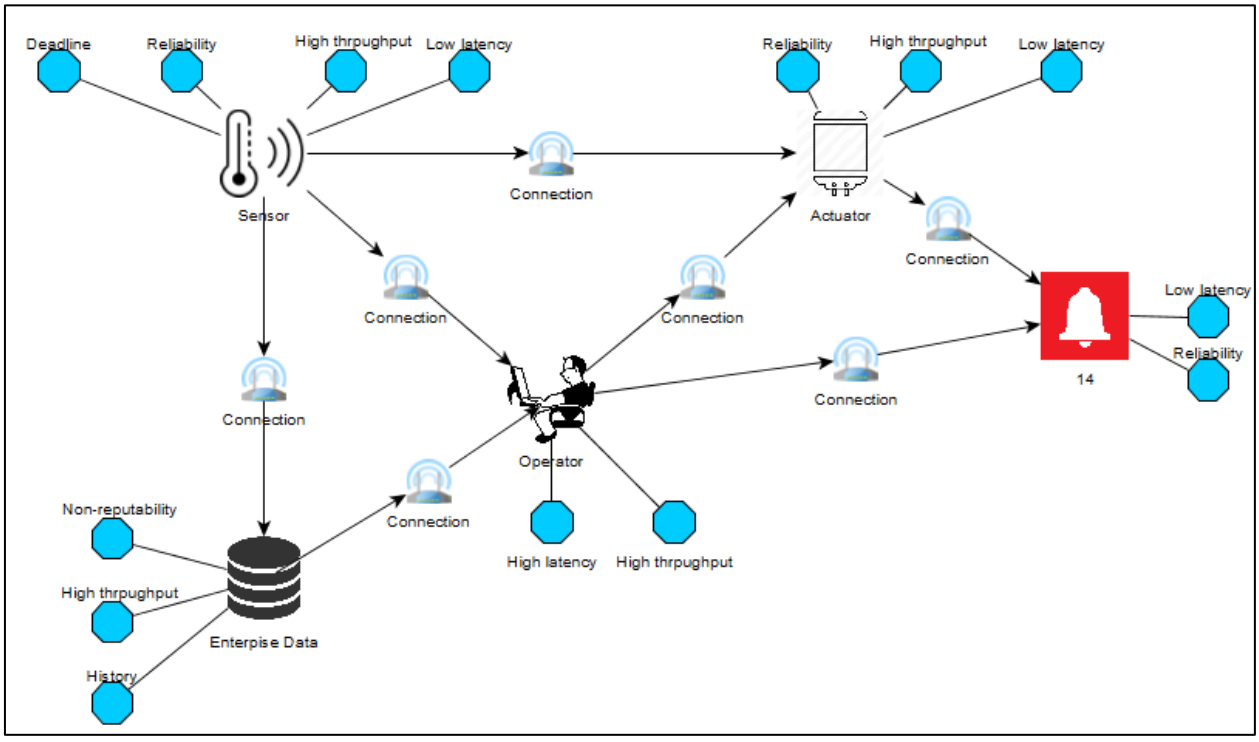
- Intuitív kezelőfelület
- Gráf képként való exportálásának lehetősége
- Részletes beállítási lehetőségek
- Csúcspontok és élek széleskörű vizuális szerkeszthetősége
- Alkalmazásspecifikus attribútumok kezelése
- Csúcspontok és élek automatikus vizuális megjelenítése az attribútumok alapján

A csomópontokhoz attribútumokat rendelve, megadható:

- A csomópont típusa (pl.: DataSource, DataSink)
- Szolgáltatásminőséget biztosító elemek (pl.: Deadline, Reliability, History)

A szerkesztő a gráfokat GraphML formátumban kezeli, ezért szerkesztés után közvetlenül lehetőség nyílik az automatikus feldolgozásra.

Az esettanulmányban ismertetett feladat a következőképpen néz ki gráfként ábrázolva:



Ábra 3-9 - Esettanulmányi példa modellje

Jelmagyarázat:

- **Ikonok** (pl.:

): Az ikonok szemléltetik a rendszerben lévő komponenseket. A jobb értelmezhetőség kedvéért a komponens működését ikonok szemléltetik.
- **Kapcsolat** (

 ): Két komponens közti kapcsolatot hivatottak jelezni, legyen az akár vezetékes vagy vezeték nélküli összeköttetés.
- **Szolgáltatásbiztonságot biztosító elemek** (pl.:

): Az ikonokhoz kapcsolódva mutatják azok követelményeit, kényszereit. (Ezek a későbbiekben a gráfcsomópontokhoz rendelt címkékként fognak megjelenni.)

3.3.3. Absztrakciós mechanizmus

3.3.3.1. Absztrakciós mechanizmus leírása

A módszertan használatához a rendszer absztrakciója [8] [20] három részre osztható. Ez a felosztás az ellenőrzési módszer miatt szükséges. Meg kell határozni a rendszer azon absztrakciós szintjét, aminél elvégezhető az automatikus tesztelési módszer.

Ábra 3-10 - Absztarkciós mechanizmus

1. **Az első absztrakciós szinten található a SysML**. A SysML-ben ábrázolhatók a komponensek, és a köztük lévő kapcsolatok. A komponensekhez és a kapcsolatokhoz felvehetők a hozzájuk tartozó követelményekből származó kényszerek, azaz a követelményaspektusokhoz megadható az adott komponensre jellemző érték. Mivel a SysML a rendszertervezési gyakorlatban igen elterjedt eszköz, ezért lehetőséget nyújt arra, hogy megfelelő kiindulási alapként szolgáljon a további rétegek számára.
2. **A második absztrakciós szinten található a rendszer gráfként való ábrázolása GraphML formátum segítségével.** Ezen absztrakciós szint szolgál az utolsó átalakítás alapjául. Az így kialakított felcímkézett csomópontok és a köztük lévő kapcsolatok XML formátumban elérhetők. Az XML formátum biztosítja a könnyű feldolgozást, amit kihasználva elvégezhető az utolsó transzformációs lépés, így elérve az utolsó absztrakciós szintet.
3. **Az legalsó absztrakciós szintet a rendszer XCSP3 kényszerekkel való felírása jelenti.** A rendszert XCSP3 formátumban ábrázolva, lehetővé teszi azt, hogy a rendszerre felírt követelmények automatikusan ellenőrizhetők legyenek.

Az absztrakciós rétegek közötti transzformációs lépések a következő módon hajthatók végre.

1. SysML → GraphML

Ebben a lépésben a SysML eszköztárából kiindulva egy GraphML leképzést kell végrehajtani. A leképzés egy GraphML szerkesztő eszköz (pl.: yEd Graph Editor) segítségével kézzel könnyen elvégezhető.

Felhasznált SysML eszköztár:

- **Association** (Asszociáció): Két komponens közti kapcsolatot fejez ki.
- **Block** (Blokk): A rendszer egy komponensét határozza meg.
- **ConstraintBlock** (Kényszer blokk): Segítségével megkötések adhatók az egyes elemekhez, a működésükre és tulajdonságaikra vonatkozóan.

Felhasznált GraphML eszköztár:

- **Node** (Csomópont): A gráf csomópontját szemlélteti.
- **Edge** (Él): Az egyes csomópontok kapcsolatát jelöli.
- **Custom properties** (Labels) (Címke): Segítségével a csomópontokhoz tulajdonságok rendelhetők.

A fentiek szerint a SysML BDD diagramon ábrázolt komponensek 1-1 csomópontnak feleltethetők meg. A SysML-ben ábrázolt eredeti összeköttetések (asszociációk) megmaradnak, azaz két kapcsolatban lévő komponenscsomópont között él fog futni a gráfban. Ezen kívül azok a csomópontok is összeköttetésre kerülnek, amiket egy kapcsolat blokk köt össze. Erre a kényszerek binarizálása miatt van szükség, így a kényszereket az transzformáció során automatikusan fel lehet majd írni a rendszerben lévő komponensek között. A SysML-ben az egyes elemekhez kapcsolt kényszerekre vonatkozó megkötések pedig a gráf adott csomópontjához kapcsolódó címkék értékeként képződnek le. A címkék nevei a vizsgált aspektusok lesznek (pl.: Technology, NonReputability, Throughput).

2. GraphML → XCSP3

Ebben a lépésben a GraphML XML formátum kerül átdolgozásra az XCSP3 formátumra, benne a rendszerre vonatkozó kényszerekkel.

GraphML eszköztár:

- **Node** (Csomópont): A gráf csomópontját szemlélteti.
- **Edge** (Él): Az egyes csomópontok kapcsolatát jelöli.
- **Custom properties** (Labels) (Címke): Segítségével a csomópontokhoz tulajdonságok rendelhetők.

XCSP3 eszköztár:

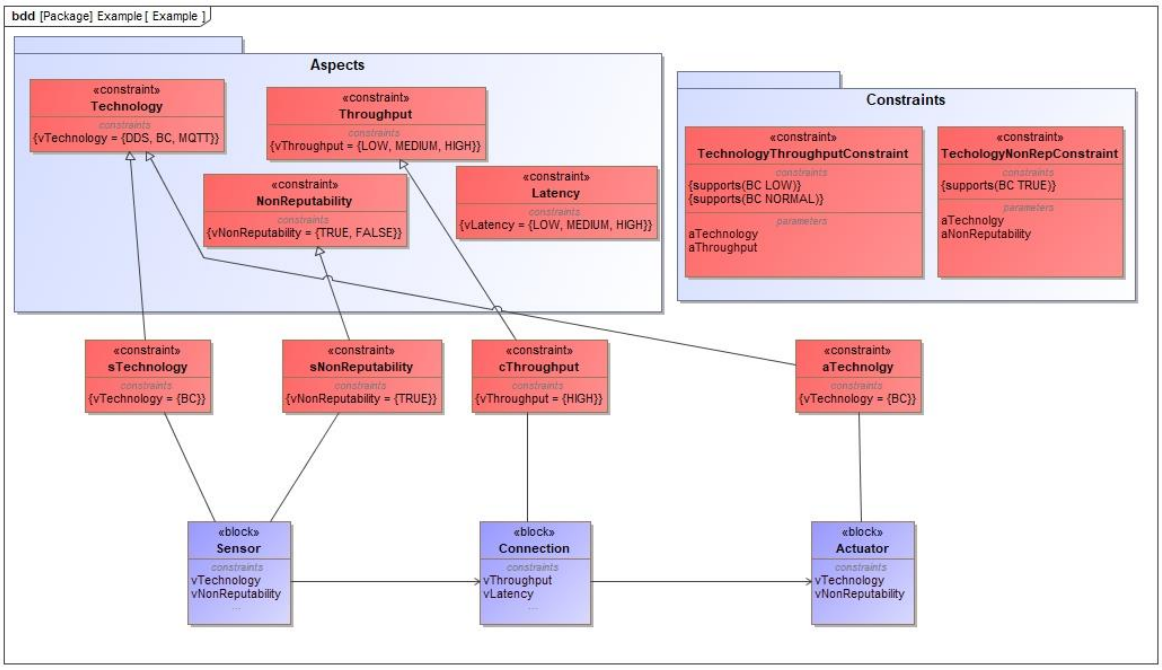
- **Variables** (Változók): A CSP változói.
  - **Arrays of Variables** (Változók tömbje): A CSP változóit tömbökbe lehet rendezni.
- **Constraints** (Kényszerek): A CSP változóira vonatkozó kényszerek.

Első lépésben a változókat kell meghatározni. A változók, az aspektusnevekkel meghatározott változótömbben foglalnak helyet. A rendszer modellezése során a lekötött változókra vonatkozó kényszerek egyértelműen meghatározhatók unáris kényszerek segítségével. A további rendszerre általános kényszereket egy előre meghatározott sablon alapján lehet generálni, ahol a változók a kényszerben szereplő kapcsolatban lévő komponensek lesznek.

3.3.3.2. Mechanizmus bemutatása

A következő példán nyomon követhető a működési folyamat.

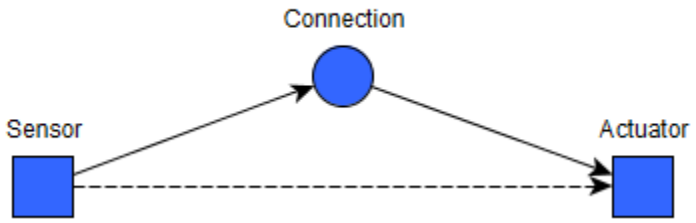
Az esettanulmányi példából kiragadott részlettel bemutatható a módszer működése. A kommunikációban (Connection) résztvevő felek egy hőmérsékletérzékelő (Sensor) és egy fűtőtest (Actuator). Az érzékelő által mért adatokat letagadhatatlan módon, Blockchain segítségével kell eljuttatni a beavatkozó számára. A mintavételezés gyakorisága miatt az átviteli csatornán nagy adatmennyiségnek kell áthaladnia.



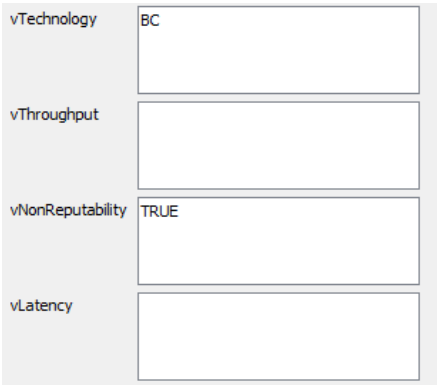
Ábra 3-11 - SysML modell

Az ábrán az *Aspects* csomagban láthatók a rendszeren vizsgált aspektusok és a hozzájuk tartozó értékkészlet. Ezen aspektusok által az egyes rendszerkomponenseinkhez kényszereket rendelhetünk, amiknél megadhatjuk a komponens adott aspektusára vonatkozó értéket (pl.: a Sensor komponensnél a használt technológia mindenképpen Blockchain legyen: *vTechnology={BC}* kényszerrel adható meg). Ezen kívül az ábrán látható egy *Constraint* csomag, ami tartalmazza a rendszerre általánosan vonatkozó megkötéseket (pl.: a *TechnologyThroughputConstraint* kimondja, hogy, ha egy csomópont szeretne Blockchaint használni, akkor a rá vonatkozó áteresztőképesség csak alacsony (LOW) és közepes (NORMAL) lehet).

Az első transzformációs lépés a SysML modell leképzése GraphML formátumra. Erre a yEd Graph Editor eszközt használtam.



Ábra 3-12 - Példa ábrázolása gráfként



Ábra 3-13 – Sensor csomópont címkéi

Az első ábrán a gráf felépítése látható, a másodikon pedig a *Sensor* csomóponthoz tartozó lekötések (címkeként) jelennek meg a SysML diagram alapján. A szaggatott él segítségével binarizálható a gráf, így az automatikus konverzió könnyebb lesz. A GraphML formátumban a fenti gráf a következőképpen jelenik meg:

```
<key attr.name="vTechnology" attr.type="string" for="node" id="d4">
  <default/>
</key>
<key attr.name="vThroughput" attr.type="string" for="node" id="d5">
  <default/>
</key>
<key attr.name="vNonReputability" attr.type="string" for="node" id="d6">
  <default/>
</key>
<key attr.name="vLatency" attr.type="string" for="node" id="d7">
  <default/>
</key>
<key attr.name="vDeadline" attr.type="string" for="node" id="d8">
  <default/>
</key>
```

Láthatóak az aspektusoknak megfelelő gráfcímkék, amelyek azonosítóval rendelkeznek, így ezekre az értékadás során hivatkozni lehet.

```
<node id="n0">
      <data key="d4"><![CDATA[BC]]></data>
      <data key="d6"><![CDATA[TRUE]]></data>
```

A címkék azonosítóit felhasználva megadhatók az adott csomóponthoz ($\boldsymbol{n_0}$) tartozó értékek ($\boldsymbol{d_4}$, azaz $\boldsymbol{vTechnology}$-ra és $\boldsymbol{d_6}$, azaz $\boldsymbol{vNonReputability}$-re).

Az utolsó lépés a XCSP3 fájl előállítása. Az XCSP3 fájlban két dolgot kell meghatározni.

1. **Változók**:

   A változókat aspektusok szerint lehet csoportosítani. Az aspektusok egy változótömböt határoznak meg, aminek a tartománya az adott aspektus tartománya. A változótömb elemeinek számát a gráf csomópontjainak száma határozza meg. Minden elem a tömbben az adott aspektus szerinti csomópontot jelöl ki (pl.: a $\boldsymbol{vTechnology}[\mathbf{1}]$ használatával az első csomópont technológiájára vonatkozó kényszer határozható meg).

```
<variables>
        <array id="vTechnology" size="[3]" type="symbolic">
                DDS MQTT BC
        </array>
        <array id="vThroughput" size="[3]" type="symbolic">
                LOW NORMAL HIGH
        </array>
        <array id="vNonReputability" size="[3]" type="symbolic">
                TRUE FALSE
        </array>
        <array id="vLatency" size="[3]" type="symbolic">
                LOW NORMAL HIGH
        </array>
</variables>
```

Látható, hogy az aspektusok szerinti 3 elemszámú ($\boldsymbol{size} = [\mathbf{3}]$) változótömbök jönnek létre. Ha $\boldsymbol{vArray}$ valamelyik változótömb, akkor a $\boldsymbol{vArray}[\mathbf{0}]$ a *Sensor*-ra, a $\boldsymbol{vArray}[\mathbf{1}]$ a *Connectionra*-ra és a $\boldsymbol{vArray}[\mathbf{2}]$ pedig az *Actuator*-ra vonatkozik.

2. Kényszerek

Az átalakítás során a kényszerek két csoportba sorolhatók.

a. Lekötött változókra vonatkozó kényszerek

```
<extension>
        <list> vTechnology[0]<list>
        <supports> BC </supports>
</extension>
<extension>
        <list> vNonReputability[0]<list>
        <supports> TRUE </supports>
</extension>
```

Az ábrán látható, hogy a *Sensor*-hoz ($[\mathbf{0}]$) a technológia és a letagadhatatlanság lekötésre került. A megoldás során ezek a változók csak ezeket az értéket vehetik fel.

b. Általánosan a rendszerre vonatkozó kényszerek

A rendszerre vonatkozó általános követelmények legenerálhatók a sablonkövetelmények szerint:

```
<!-- BC-t használva csak LOW vagy NORMAL lehet a TP -->
<extension>
        <list> vTechnology[i] vThroughput[k] </list>
        <supports> BC LOW </supports>
        <supports> BC NORMAL </supports>
</extension>
```

Ez azt fejezi ki, hogy $\boldsymbol{i}$ csomóponton Blockchaint használva $\boldsymbol{k}$ csomóponton a késleltetés csak alacsony és közepes lehet.

A gráfból kinyert szomszédsági mátrix alapján az $\boldsymbol{i}$ és $\boldsymbol{k}$ iterátorok helyére megadhatók azok a csomópontok, amikre a kényszer vonatkozik.

```
<extension>
        <list> vTechnology[0] vThroughput[2] </list>
        <supports> BC LOW </supports>
        <supports> BC NORMAL </supports>
</extension>
<extension>
        <list> vTechnology[2] vThroughput[0] </list>
        <supports> BC LOW </supports>
        <supports> BC NORMAL </supports>
</extension>
```

Az ábrán látható, hogy a *Sensor* és az *Actuator* közti kapcsolatra szeretnénk megkötést tenni a használt technológia és az áteresztőképességet illetően.

Az így kialakított XCSP3 fájl használható lesz a CSP megoldó bemeneteként, így a rendszerre vonatkozó CSP megoldható, ezzel ellenőrizhető a rendszer szolgáltatásbiztonsága.

Fontos megjegyezni, hogy a kényszersablonokat valamilyen formában kézzel kell előállítani, mivel a GraphML leképzése nem lenne megfelelő.

Algoritmus az GraphML átalakítására XCSP3-ra

Bemenetek:

- $\boldsymbol{V} = \{\boldsymbol{aspect_1}[\boldsymbol{N}] = \{\boldsymbol{D_1}\}, \boldsymbol{aspect_2}[\boldsymbol{N}] = \{\boldsymbol{D_2}\}, \dots\}$, a CSP $\boldsymbol{N}$ hosszú változótömbjei, ahol $\boldsymbol{ascpect_i}[\boldsymbol{N}]$, az $\boldsymbol{i}$. aspektushoz tartozó $\boldsymbol{N}$ csomópontra vonatkozó kényszerváltozó, $\boldsymbol{D_i}$ pedig az adott aspektushoz tartozó domén.
- $\boldsymbol{n(V)}$ – Az aspektusok száma.

- $\boldsymbol{G}$ gráf – a rendszer gráfként való ábrázolása
- $\forall\, \boldsymbol{v_i} \in \boldsymbol{V(G)}$: $\boldsymbol{e_i} \rightarrow \boldsymbol{labels}\{\boldsymbol{aspect_1}, \boldsymbol{aspect_2}, \dots, \boldsymbol{aspect_{n(V)}}\}$, az $\boldsymbol{i}$. csomóponthoz tartozik egy $\boldsymbol{n(V)}$ elemszámú halmaz, aminek elemei az adott csomóponthoz tartozó címkék.
- $\boldsymbol{N} = \boldsymbol{v(G)}$ – a gráf csúcsainak száma

- $\forall\, \boldsymbol{e_i} \in \boldsymbol{E(G)}, \boldsymbol{e_i}[\boldsymbol{label_k}] \rightarrow \boldsymbol{V_k}[\boldsymbol{i}], \boldsymbol{k} = \boldsymbol{1} \dots \boldsymbol{n(V)}$, azaz minden csomóponthoz tartozó címke megfeltethető az egyes változótömbök elemeinek.

- $\boldsymbol{C} = \{\boldsymbol{constraint_i}(\boldsymbol{V_k}[\boldsymbol{l}], \boldsymbol{V_n}[\boldsymbol{m}], \dots), \dots\}$ – A kényszerek halmaza. A kényszer paraméterei a kényszerben részt vevő változókat jelölik.

- $\boldsymbol{V_{xcsp3}}$ – Az XCSP3 kényszerváltozók halmaza.
- $\boldsymbol{U_{xcsp3}}$ – A lekötött értékkel rendelkező változók halmaza.
- $\boldsymbol{C_{xcsp3}}$ – A további kényszerek, amik a rendszerre vonatkoznak.

Algoritmus

- Változók

```
for each v in V do
    V_xcsp3 = V_xcsp3 ∪ v
endfor
```

Azaz, a XCSP3 fájlhoz hozzáadódnak a változótömbök deklarációi.

- Lekötött változók

```
for each v in v(G) do
   for each l in v.labels do
      if l is not empty,
      then
         U_xcsp3 = U_xcsp3 ∪ l
      endif
   endfor
endfor
```

Azaz, azok a változók, amikhez kényszer lett lekötve, unáris kényszerként füzödnek hozzá az XCSP3 fájlhoz.

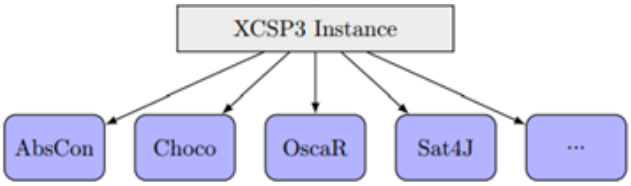
- Kényszerek

```
for each (v_i, v_j) in e(G), i ≠ j} do
   for each c in C do
      C_xcsp3 = C_xcsp3 ∪ c(v_i, v_j, …)
   endfor
endfor
```

A gráfban lévő minden élre (a kapcsolatok és a komponensek között, valamint a kapcsolattal összekötött komponensek között is) fel kell venni a rendszerre vonatkozó kényszereket, így sablonban meghatározott kényszerek vizsgálva lesznek az egész rendszerre nézve.

3.3.4. Modell ellenőrzése

A szolgáltatásbiztonság ellenőrzésének utolsó lépése maga a kényszerkielégítési probléma megoldása.



Ábra 3-14 - XCSP3 megoldása

Az XCSP3 fájl elkészítése után a megoldó algoritmusok használhatók a CSP megoldásainak meghatározására, ezáltal a probléma megoldásával válasz kapható a rendszer helyességére vonatkozóan. Megkapható egy vagy az összes megoldás és az is kiderül, ha az adott megkötésekkel nem működik a rendszer. A CSP megoldására használhatók előre elkészített programok, amik végre tudják hajtani a megfelelő algoritmusokat a problémán.

3.3.4.1. CSP megoldók

A CSP megoldók (solvers) olyan programok, amiket kényszerkielégítési problémák megoldására fejlesztettek. Általában egy API-t biztosítanak a CSP modellezésére és a modell alapján a megoldás meghatározására. A megoldók minősége és hatékonysága széles skálán mozog, a megvalósításuktól függően.

Az XCSP3-t támogató megoldók:

- **AbsCon**[9], általános célú CSP megoldó
- **Choco**[10], nyílt forráskódú Java könyvtár kényszerprogramozáshoz
- **Concrete**[11], egy CP rendszer Scala alapon
- **OscaR**[12], Operációkutatás Scala alapon
- **Sat4J**[13], Logikai kielégítési és optimalizációs Java könyvtár

Az XCSP által 2017-ben kiírt megoldó versenyen[14] a felsorolt programok mind elindultak. A cél CSP és COP (Constraint Optimalization Problem – kényszeroptimalizálási probléma) feladatok megoldása volt. A versenyzőket több szempont alapján értékelteltél. Figyelembe vették a megoldási időket, a megoldott és a hibás megoldást adó feladatok számát is. A feladatok nehézsége változatos volt, az egyszerű pár kényszert és változót tartalmazó feladattól a több százezer kényszert és változót tartalmazó feladatokig.

A versenyben elért eredményei[15] alapján a Choco megoldó megfelelő eszköznek bizonyul a CPS ellenőrzésére.

Category "decision problem" (CSP)

Solver Name	#benchs	Solved	UNSAT.	OPT	SAT	SAT (subopt.)	SAT (timeout)	SAT (out of mem.)	UNKNOWN	UNKNOWN (no support)	UNKNOWN (timeout)	UNKNOWN (out of mem.)
AbsCon-basic *2017-06-11* (complete)	510	368 (72.16%)	102 (20%)	0 (0%)	266 (52.16%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	1 (0.2%)	127 (24.9%)	0 (0%)
BTD *2017-07-25* (complete)	510	240 (47.06%)	82 (16.08%)	0 (0%)	158 (30.98%)	0 (0%)	0 (0%)	0 (0%)	11 (2.16%)	151 (29.61%)	101 (19.8%)	5 (0.98%)
BTD *2017-08-10* (complete)	510	241 (47.25%)	82 (16.08%)	0 (0%)	159 (31.18%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	151 (29.61%)	112 (21.96%)	5 (0.98%)
choco-solver *4.0.5 par (2017-08-09)* (complete)	510	393 (77.06%)	111 (21.76%)	0 (0%)	282 (55.29%)	0 (0%)	0 (0%)	0 (0%)	15 (2.94%)	0 (0%)	100 (19.61%)	0 (0%)
choco-solver *4.0.5 par (2017-08-18)* (complete)	510	396 (77.65%)	109 (21.37%)	0 (0%)	287 (56.27%)	0 (0%)	0 (0%)	0 (0%)	17 (3.33%)	0 (0%)	97 (19.02%)	0 (0%)

Táblázat 3-1 - CSP verseny eredmények

Az ábrán látható, hogy a versenyen résztvevő megoldók egy része (köztük a Chocoval) hogyan szerepelt. Látható, hogy hány feladatot sikerült megoldaniuk (Solved), ezen belül hány feladatot jelöltek meg megoldhatatlannak (UNSAT) és megoldottnak (SAT). A többi oszlopban a sikertelen megoldások szerepelnek (UNKOWN (időtúllépés, kevés memória).

[9] http://www.cril.univ-artois.fr/~lecoutre/software.html
[10] http://choco-solver.org/
[11] https://github.com/concrete-cp/concrete
[12] https://bitbucket.org/oscarlib/oscar/wiki/Home
[13] http://www.sat4j.org/
[14] http://www.cril.univ-artois.fr/XCSP17/
[15] https://www.cril.univ-artois.fr/CompetitionXCSP17/results/globalbysolver.php?idev=88&det=2

3.3.4.1.1. Choco

Solver

A Choco [7] [16] egy ingyenes, nyílt forrású Java 8 könyvtár, amit kényszerprogramozásra fejlesztettek ki. A felhasználó deklaratív módon meghatározhatja a kényszereket, amiknek minden megoldásban teljesülniük kell. A problémát alternáló kényszerszűrő algoritmusokkal és keresési mechanizmusokkal oldja meg a könyvtár.

A Choco 4 támogatja:

- Változatos változótípusokat (egész, logikai, valós, halamaz)
- Többféle kényszert (pl.: alldifferent, count, nvalues)
- Keresési beállítások módosítását (pl.: saját keresési algoritmus megadása)
- Konfliktusok esetén történő működést (pl.: visszaugrás, dinamikus backtrack)

A következőkben látható lesz, hogy a Choco alkalmas a CPS szolgáltatásbiztonságának ellenőrzésére, mivel képes jóval nagyobb bemenettel rendelkező problémák megoldására. A versenyen megoldott feladatok adatai és a probléma megoldásának sikeressége látható a táblázatokon. Olvasható a probléma neve (*Name*), a megoldó eredménye (*Best restult obtained on this benchmark*), a változókra (*Number of variables*), kényszerekre (*Number of constraints*) és doménekre (*Number of domains*) vonatkozó információk, ezeken kívül a megoldás ideje is fel van tűntetve a táblázatban (*Best CPU time*).

A QueenAttacking probléma megoldásainak meghatározása sikeres volt a 60 változóval és 2000 kényszerrel.

Name	QueenAttacking/QueenAttacking-xcsp2-s1/ QueenAttacking-08_X2.xml
MD5SUM	01373ebb4a146089a71bb04c36ed4f3a
Bench Category	CSP (decision problem)
Best result obtained on this benchmark	SAT
Best value of the objective obtained on this benchmark	
Best CPU time to get the best result obtained on this benchmark	2118.48
Satisfiable	
(Un)Satisfiability was proved	
Number of variables	65
Number of constraints	2034
Number of domains	2

Táblázat 3-2 - QueenAttacking probléma megoldásának információi

Megvizsgálva a PropStress problémára adott megoldást, látható, hogy egy megoldással nem rendelkező kényszerkielégítési problémára 500 változó és 30000 kényszer esetén is sikerült az ellenőrzés.

Name	PropStress/PropStress-m1-s1/ PropStress-0250.xml
MD5SUM	fc49515a3b3121bae6777db9cebdf786
Bench Category	CSP (decision problem)
Best result obtained on this benchmark	UNSAT
Best value of the objective obtained on this benchmark	
Best CPU time to get the best result obtained on this benchmark	109.141
Satisfiable	
(Un)Satisfiability was proved	
Number of variables	502
Number of constraints	31876
Number of domains	1

Táblázat 3-3 - PropStress probléma megoldásának információi

A Choco által megoldott problémák alapján látható, hogy a CPS-ben felmerült kényszerkielégítési problémák is időben megoldhatók a Choco használatával.

A CSP, annyi változót tartalmaz, mint a CPS rendszerben előforduló komponensek száma és a vizsgált aspektusok szorzata. A kényszerek száma függ a rendszerben lévő összeköttetések számától és a rendszerre vonatkozó általános követelményektől, a domének száma pedig az aspektusok számával egyezik meg. Ezekkel a korlátos mennyiségekkel a CSP egy valódi rendszerre nézve is megoldható lesz.

Parser

A Choco fejlesztői biztosítanak egy Parsert[16], aminek segítségével különböző formátumú CSP modelleket lehet importálni a megoldóhoz. Támogatja az XCSP3 formátumot is.

Ezzel a módszerrel a CPS kényszereit leíró XCSP3 fájl alapján ellenőrizhető a rendszer és megoldás/megoldások adhatók.

3.4. Követhetőség biztosítási az absztrakció során

A mérnöki gyakorlatban elengedhetetlen a követelmények nyomonkövethetőségének (Traceability) biztosítása. Erre általában lehetőséget biztosítanak a fejlesztés során használt tervező eszközök (IBM Doors, ReqIF Studio). A rendszer ellenőrzése és későbbi karbantartása során így lehetőség van a hibák könnyeb feltárására és javítására. A funkciók módosítását, bővítését is könnyebbé teszik.

A módszerben több átalakítási lépés történik, melyek folyamán a rendszert különböző módokon kell ábrázolni. Az átalakítások következtében a komponensek tulajdonságai (helye, funkciója a rendszerben) elveszhetnek, ezért a nyomonkövethetőséget ebben az esetben is biztosítani kell.

A transzformációs lépések között lehetőséget kell biztosítani a komponensek és a kényszerek követhetőségére, így hiba esetén lehetséges a hibás rész beazonosítása, ezért a javítás könnyebb lehet.

A kidolgozott transzformációs modellben látható, hogy a tervezésénél fogva biztosítja a követhetőséget. Gyakorlatban a SysML és a GraphML közti leképzési transzformációban a nyomonkövethetőség a komponensek nevével és a hozzá tartozó megkötésekkel valósul meg. A GraphML és az XCSP között az gráf csomópontjainak azonosítói nyújtanak segítséget a nyomonkövethetőség biztosításához.

4. Összefoglalás

Az ismertetett módszerek lehetővé teszik a CPS egész tervezési folyamatán átívelő algoritmikus ellenőrzést. A SysML integrálhatóság lehetővé teszi, hogy valódi projektekben is használható legyen az eljárás.

A módszer független a felhasznált technológiáktól, ezért egy új eszköz megjelenése vagy egy régi támogatásának megszűnése sem okoz gondot.

A tervezési folyamatban észlelt hibák jelentősen lecsökkenthetik a projekt költségeit és a tervezési időt.

[16] https://github.com/chocoteam/choco-parsers

4.1. Továbbfejlesztési lehetőségek

Algoritmikus továbbfejlesztési lehetőségek

A megoldás során a keresés a CSP egy vagy az összes megoldását adja vissza. Egy megoldás esetén nem lehet biztosan tudni, hogy az az adott megoldás tényleg a legmegfelelőbb-e, akár műszaki (pl.: homogén rendszer megvalósítása egyszerűbb lehet), akár a rendszer költségének (pl.: egyes komponensek részköltsége befolyásolja a teljes költséget) szempontjából. Az optimális megoldás kiválasztása igény lehet a felhasználó szempontjából. Ehhez felhasználhatók prioritási kényszerek (a megoldás szempontjából előnyben részesített értékek prioritást élveznek), amikkel biztosítani lehet, hogy a megoldás keresése során az optimális megoldás szülessen meg. Ezen kívül az is befolyásolja a megoldás minőségét és idejét, hogy a kényszerváltozók milyen sorrendben lesznek lekötve (pl.: tervezési minta használata estén a komponens belső változóit érdemes először lekötni, mivel azok kihatással lehetnek a külső kényszerekre). Ezekkel a fejlesztésekkel növelhető a hatékonyság és finomíthatók a felhasználói igények.

Technológiai továbbfejlesztési lehetőségek

- **IBM Rhapsody intergrálhatóság**: A CPS SysML modellezése elvégezhető a Rhapsody használatával. A Rhapsody támogatja a DDS használatát is, így ezek integrálhatósága megoldható.
- **Hiba helyének pontos beazonosítása**: Inkonzisztencia esetén a felhasználónak pontos visszajelzést lehet adni nem csak a hiba típusáról, hanem a helyéről is (melyik csomópontok között nem teljesülnek a kényszerek) ezzel még jobban csökkenthető a tervezési idő.
- **Fellépő hiba esetén javaslatok a javításra**: Ha az ellenőrzés során kiderül, hogy a rendszer ellentmondó követelmények miatt nem megvalósítható, akkor a hiba típusától függően javaslat adható annak javítására (pl.: ha a hiba az alacsony áteresztőképesség miatt keletkezik, akkor a probléma megoldható lehet szűrő beiktatásával, ami kevesebb adatot enged át a kapcsolaton).
- **Keretrendszer implementálása**: Az eljárás implementálása folyik. Segítségével valós projektekben is lehet alkalmazni a rendszer ellenőrzésére vonatkozó módszertart.
- **Ellenőrzésre való módszertan**: Miután a szóba jövő modellek a magas tervezési szint miatt nem túlzottan nagyok, ezért várhatóan skálázhatósági gond nem merül fel.

Irodalomjegyzék

[1] Avizienis, A., Fellow, IEEE, Laprie, J.-C., Randell, B., & Landwehr, C. (2004). Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING*. Forrás: Basic Concepts and Taxonomy of Dependable and Secure Computing: https://www.nasa.gov/pdf/636745main_day_3-algirdas_avizienis.pdf

[2] Barták, R. (1998). *ON-LINE GUIDE TO CONSTRAINT PROGRAMMING.* Forrás: http://kti.mff.cuni.cz/~bartak/constraints/

[3] Barták, R. (1999). *Constraint Programming: In Pursuit of the Holy Grail.*

[4] Brandes, U., Eiglsperger, M., Lerner, J., & Pich, C. (2013). Graph Markup Language (GraphML). In R. Tamassia, *Handbook of Graph Drawing and Visualization* (old.: 517-541).

[5] HAKIRI, A., BERTHOU, P., Gokhale, A., Schmidt, D., & Thierry, G. (2013). Forrás: Supporting End-to-end Scalability and Real-time Event Dissemination in the OMG Data Distribution Service over Wide Area Networks: https://www.dre.vanderbilt.edu/~schmidt/PDF/elsarticle-journal.pdf

[6] *Industrial Internet Consortium.* (2017. July 19). Forrás: The Industrial Internet of Things Vocabulary: http://www.iiconsortium.org/pdf/IIC_Vocab_Technical_Report_2.0.pdf

[7] Jussien, N., Rochart, G., & Lorca, X. (2010. May 12). Forrás: Choco: an Open Source Java Constraint Programming Library: http://hal.univ-nantes.fr/hal-00483090/

[8] László, G., István, M., Szilárd, B., & András, P. (2016). *MDD-Based Design, Configuration, and Monitoring of Resilient Cyber-Physical Systems.*
[9] Murphy, B. (2017. January 31). *RTI Blog*. Forrás: 2nd Version of the Industrial Internet Reference Architecture is Out with Layered Databus: http://blogs.rti.com/2017/01/31/2nd-version-of-the-industrial-internet-reference-architecture-is-out-with-layered-databus
[10] *NIST CPS Public Working Group.* (2016. May). Forrás: CPS PWG Cyber-Physical Systems (CPS) Framework Release 1.0: https://s3.amazonaws.com/nist-sgcps/cpspwg/files/pwgglobal/CPS_PWG_Framework_for_Cyber_Physical_Systems_Release_1_0Final.pdf
[11] *OASIS MQTT.* (2014. October 29). Forrás: MQTT Version 3.1.1: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf
[12] *Object Management Group*. (dátum nélk.). Forrás: OMG SysML: http://www.omgsysml.org/
[13] Pataricza, A., Gönczy, L., Kövi, A., & Szatmári, Z. (2011). *A Methodology for Standards-Driven Metamodel Fusion.*
[14] Pérez, H., & Gutiérrez, J. (2015). Modeling the QoS parameters of DDS for event-driven real-time applications. *Journal of Systems and Software Volume 104*, old.: 126-140.
[15] *PrismTech.* (2013. November). Forrás: Messaging Technologies for the Industrial Internet and the Internet of Things: http://www.prismtech.com/sites/default/files/documents/MessagingComparsionNov2013USROW_vfinal.pdf
[16] Prud'homme, C., Fages, J.-G., & Lorca, X. (2016. February 9). *Choco Solver 4*. Forrás: Choco Solver Documentation: http://www.choco-solver.org
[17] Soares, M., & Vrancken, J. (2008). Model-Driven User Requirements Specification using SysML. *JOURNAL OF SOFTWARE VOL. 3, NO. 6*, 57-68.
[18] *The GraphML File Format*. (dátum nélk.). Forrás: http://graphml.graphdrawing.org/
[19] Tsang, E. (1996). *Foundations of Constraint Satisfaction: The Classic Text.*
[20] Varró, D., & Pataricza, A. (2003. October 1). VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML (The Mathematics of Metamodeling is Metamodeling Mathematics). *Software and Systems Modeling*, 187-210.
[21] *XCSP3*. (dátum nélk.). Forrás: http://xcsp.org/
[22] *XCSP3*. (dátum nélk.). Forrás: XCSP3-Java-Tools: https://github.com/xcsp3team/XCSP3-Java-Tools
[23] *yWorks the diagramming company*. (dátum nélk.). Forrás: yEd Graph Editor: https://www.yworks.com/products/yed
[24] *https://www.pubnub.com/wp-content/uploads/2017/06/edge-computing-diagram-1024x512.png*
[25] *https://rti.wistia.com/medias/8ma88ry3mw?embedType=async&videoFoam=true&videoWidth=640*
[26] Mohan, N., & Kangasharju, J. (n.d.). Edge-Fog cloud: A distributed cloud for Internet of Things computations.
[27] Cloud Standards Customer Council http://www.cloud-council.org/about-us.htm
[28] Cloud Customer Architecture for IoT http://www.cloud-council.org/deliverables/CSCC-Cloud-Customer-Architecture-for-IoT.pdf
[29] OpenFog Consortium https://www.openfogconsortium.org
[30] *RTI Connext -Core Libraries and Utilities* http://community.rti.com/rti-doc/500/ndds.5.0.0/doc/pdf_html/RTI_CoreLibrariesAndUtilities_GettingStarted.html
[31] *Barnett, D. (2013, May). Retrieved from Comparison of MQTT and DDS as M2M Protocols for the Internet of Things: https://www.slideshare.net/RealTimeInnovations/comparison-of-mqtt-and-dds-as-m2m-protocols-for-the-internet-of-things*